.40377

National Library of Canada / Bibliothèque nationale du Canada

CANADIAN THESES ON MICROFICHE / THÈSES CANADIENNES SUR MICROFICHE

NAME OF AUTHOR/*NOM DE L'AUTEUR* Andrew Brennek

TITLE OF THESIS/*TITRE DE LA THÈSE* DISCO: A Distributed Computer Control Software System for Engineering Processes.

UNIVERSITY/*UNIVERSITÉ* University of Alberta.

DEGREE FOR WHICH THESIS WAS PRESENTED/*GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* MSc. in Process Control

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE GRADE* 1979

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* Prof. D. Grant Fisher

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/*DATÉ* 11 XII 1978 SIGNED/*SIGNÉ* [signature]

PERMANENT ADDRESS/*RÉSIDENCE FIXE* 10619 50 st,
Edmonton, Alberta.

NL-91 (3-74)

National Library of Canada

Cataloguing Branch
Canadian Theses Division

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada

Direction du catalogage
Division des thèses canadiennes

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

NL-339 (3/77)

THE UNIVERSITY OF ALBERTA

DISCO: A DISTRIBUTED COMPUTER CONTROL SOFTWARE SYSTEM FOR ENGINEERING PROCESSES

by

© ANDREW BRENNEK

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

IN

PROCESS CONTROL

DEPARTMENT OF CHEMICAL ENGINEERING

EDMONTON, ALBERTA

SPRING 1979

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled

DISCO: A Distributed Computer Control Software System for Engineering Processes

submitted by Andrew Brennek

in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

IN PROCESS CONTROL

.....
Supervisor

.....

.....

Date... December 5, 1978 ....

ABSTRACT

This thesis describes the design and implementation of a DIstributed Supervision and COntrol (DISCO) system. DISCO is a software suite for the control and supervision of engineering processes by a distributed network of digital computers. The programs have been designed for operation by a team possessing skills commonly found in a modern, industrial process control group.

DISCO has been designed as a very flexible, general purpose control system. A modular structure has been incorporated to ensure that the system can be tailored and expanded as required for specific applications and differing operational environments. This design ensures that DISCO will not only execute conventional 'feedback loop' algorithms but will also provide the means for implementing more sophisticated control schemes. The advanced category includes multivariable, adaptive and decoupling control. All of these can be processed in true, table-driven DDC mode under the direction of the DISCO executive. Software modules used to implement advanced schemes used in a university research environment can, however, be omitted from an industrial implementation. The operation of the modules constituting the reduced system would be unchanged.

Supervisory programs may be incorporated in the DISCO system where the need for task-driven rather than table-

driven software arises. Programs of this type could be implemented to perform operations such as process optimisation or parameter estimation. The supervisory software would exist in software layers hierarchially above those of the DDC system but would, nevertheless, be allowed access to the DDC data-base.

The thesis discusses the classes of problems handled by DISCO and important objectives in the system synthesis. The steps taken in choosing a process data-base structure which takes into account both the needs of the application and the tools of the implementation have been carefully documented. Testing of an experimental, single-processor, in-memory version of DISCO is described and implications with respect to a complete implementation are drawn from these first experiences.

In carrying out the project every effort has been made to fully exploit modern distributed computer processing technology, improved hardware speeds and high-level real-time programming. It is concluded that the use of state-of-the-art tools and techniques has yielded a more flexible, reliable and hence, potentially more powerful system than any commercially available. These objectives have been achieved without suffering limited throughput or degraded response times.

The DISCO system is undergoing continued development in the Department of Chemical Engineering. This thesis describes aspects of the implementation to which active

effort is presently directed and discusses the main areas in which work is required to achieve a full realisation of the design. It is expected that this work will, in the future, refine DISCO to a stage at which it constitutes a realistic prototype for a commercial computer control scheme.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. D. Grant Fisher for his patient and stimulating guidance throughout the project. Sincere thanks are also extended to the National Research Council, the Province of Alberta and Imperial Oil Limited for their financial support of the research project. The author's wife, Barbara, is gratefully credited with painstaking formatting and typing of the manuscript.

The staff of the DACS centre at the Department of Chemical Engineering deserve particular mention for their helpful technical assistance at various stages of the DISCO implementation. The author feels especially indebted to Mr. Vladimir Berka, the DACS centre manager, for his encouragement and his very significant technical contribution to the shaping of the DISCO project.

TABLE OF CONTENTS

CHAPTER		PAGE
1.	INTRODUCTION	1
	1.1 Introduction to the Project	1
	1.2 DISCO in the Context of Other Work in the Field	4
	1.3 Documentation of the Thesis Project ...	9
2.	PROJECT OVERVIEW	14
	2.1 Principle Research Interests	14
	2.2 The Process Activity and the Activity Record	15
	2.3 Process Services Provided by DISCO	21
3.	SCOPE AND CAPABILITIES OF DISCO	27
	3.1 Problems Handled by DISCO	27
	3.2 Design Objectives	28
	3.2.1 Objectives with Respect to Conventional DDC Capability ...	30
	3.2.2 Distributed Computer Network Operation	38
	3.2.3 Objectives with Respect to Support of Advanced Control ...	43
	3.3 Tools and Media of the Implementation .	43
	3.3.1 Computer System Hardware	45
	3.3.2 Network Software	46
	3.3.3 The Programming Language of the Implementation	48
	3.3.4 Services Provided by the Real-Time Operating System	50
4.	DESIGN OF THE DATA-BASE	52
	4.1 Table-Driven Processing	52

TABLE OF CONTENTS....continued

CHAPTER		PAGE
	4.2 Choice of a Data-Base Structure	55
	4.3 Compromises Made in Meeting the Design Objectives	57
	4.4 The Data-Base Structure	62
	4.4.1 The Activity Record	63
	4.4.2 Subsets of the Activity Record	65
	4.4.3 Organisation of the Data-Base .	67
	4.4.4 Distribution of the Data Base .	71
	4.5 A Comparison with Data-Base Design Procedures in a Different Working Environment	73
5.	EXPERIMENTAL IMPLEMENTATION AND EVALUATION OF DISCO	78
	5.1 Introduction to the Implementation	78
	5.1.1 Software Project Planning	81
	5.2 Implementation of Interactive Building of a DISCO Data-Base	83
	5.2.1 The Need for Software Support .	83
	5.2.2 Different Sources and Different Types of Information	87
	5.2.3 Introduction to the Operation of Data-Base Building	90
	5.2.4 DOPE the Data-Base OPerations Executive	95
	5.2.5 A Partial Implementation	95
	5.3 Real-Time Programs in the DISCO Suite .	99
	5.3.1 The DISCO Executive	99
	5.3.2 Supervisory Programs	102
	5.3.3 ACCES (The Data-Base Communication Processor)	103

TABLE OF CONTENTS....continued

CHAPTER			PAGE
	5.3.4	Integrity Measures Associated with Real-Time Data-Base Manipulation	105
6.	EXAMPLES OF DISCO APPLICATIONS		113
	6.1 Introduction to the Chapter		113
	6.2 The Example Application		113
	6.3 Design and Implementation of DISCO Segments		120
	6.4 An Industrial Advanced Control Application		127
	6.4.1	Description of the Example	127
	6.4.2	A Proposed DISCO Activity Configuration	132
7.	CONCLUSIONS AND CONTINUATION OF THE PROJECT		138
	7.1 Conclusions of the Thesis Project		138
	7.2 The Continuing Project		142
	7.2.1	Future Work in Expanding the Data-Base Building Utilities ..	142
	7.2.2	Future Work in Development of Process Operator Console Support	145
	7.2.3	Future Work in Development of New Segment Types	146
	7.2.4	Future Work in the Development of the DISCO Real-Time Software	147
	7.2.5	Future Work in Expansion of DISCO to Multiprocessor Operation	148
REFERENCES			150
APPENDIX A			163

TABLE OF CONTENTS....continued

CHAPTER		PAGE
	A.1 Introduction to Appendix A	163
APPENDIX B		170
	B.1 Introduction to Appendix B	170

LIST OF TABLES

TABLE		PAGE
1.1	Summary critique of Honeywell TDC 2000 properties	7
1.2	Bibliography of documentation for DISCO modules implemented as part of the thesis project	12
2.1	A representative selection of possible segment processors	19
3.1	Summary of objectives for conventional DDC capability	31
3.2	Summary of objectives for the operation of DISCO on a distributed network of computers	39
3.3	Summary of objectives for advanced control capability	44
3.4	Features of the real-time operating system which have been used to advantage in DISCO	51
4.1	Typical process services implemented by table-driven software in DISCO	53
4.2	Criteria which affect choice of a structure in data-base design	56
4.3	Summary of data-base design features chosen to satisfy the objectives of Chapter 3	58
4.4	The activity header	64
4.5	Performance requirements of the CERN control scheme and of a general DISCO scheme	75
4.6	Design features of the CERN control scheme and of a general DISCO scheme	75

LIST OF TABLES....continued

TABLE		PAGE
5.1	Features of the part-implementation of DISCO undertaken in the thesis project	80
5.2	Summary of main areas of future work in implementing a complete DISCO system	80
5.3	Services provided by the data-base operations executive	97
5.4	Interim measures for features of the data-base building utilities not implemented as part of the thesis project	98
5.5	Probability of processing an activity record corrupted by bad data	107
6.1	The sequence of control options excercised by the program DEMO	116
6.2	Input segment format designed and implemented for operation in the DISCO data-base	121
6.3	Control segment format designed and implemented for operation in the DISCO data-base	122
6.4	Output segment format designed and implemented for operation in the DISCO data-base	123
7.1	Distinguishing features of the DISCO data-base design	140
7.2	Distinguishing features of the DISCO data-base building utilities	141
7.3	Distinguishing features of the DISCO real-time software	143

LIST OF TABLES....continued

TABLE		PAGE
B.1	Format of the proposed arithmetic segment .	173
B.2	Example of use of the proposed arithmetic segment in the example of section 6.4	174

LIST OF FIGURES

FIGURE		PAGE
1.1	Documentation of DISCO modules implemented as part of the thesis project	13
2.1	An example of possible applications of the main subsets of a DISCO data-base in a refinery control scheme	18
2.2	Interaction of DISCO program modules with the data-base	22
2.3	Use of DISCO program modules in a distributed computer network implementation of DISCO ...	24
3.1	Direct digital control of a single-variable feedback loop	33
3.2	Special purpose control room peripherals which must be handled by control software .	34
3.3	Schemmatic illustration of a typical application of computerised telemetry and telecontrol in a water distribution system	36
3.4	Schemmatic illustration of a typical laboratory digital control application	37
3.5	Schemmatic representation of the distributed network of digital computers operated by the DACS centre	41
4.1	Schemmatic representation of subsets of an activity record	66
4.2	Layout of the DISCO data-base	68
4.3	Schemmatic representation of DISCO Input/Output operations	70

FIGURE		PAGE
5.1a	Schemmatic representation of the data format of a typical DISCO segment	84
5.1b	Schemmatic representation of the data format of a typical DISCO I/O table	84
5.2	Information required to build and load a data-base	88
5.3	Schemmatic representation of data-base building operations as governed by the data-base operations executive	91
5.4	Menu structure of the data-base operations executive	96
5.5	Top-down processing of the ART	101
5.6	Schemmatic representation of error recovery mechanisms which may be used to ensure the integrity of the distributed system	111
6.1	Schemmatic diagram of the instrumentation evaluation rig used for an example application	114
6.2	Response of the Tank 2 level to the control options excercised by program DEMO	118
6.3	Example of interacting reboilers taken from Shinskey [*]	129
6.4	The decoupling control scheme proposed by Shinskey [*]	131
6.5	A proposed activity configuration for the decoupling control scheme	135

FIGURE		PAGE
A.1a	A listing of output from the memory-image builder 'MBILD' for the example application of Chapter 6	164
A.1b	A listing of output from the memory-image builder 'MBILD' for the example application of Chapter 6 ...contd.	165
A.1c	A listing of output from the memory-image builder 'MBILD' for the example application of Chapter 6 ...contd.	166
A.1d	A listing of output from the memory-image builder 'MBILD' for the example application of Chapter 6 ...contd.	167
A.1e	A listing of output from the memory-image builder 'MBILD' for the example application of Chapter 6 ...contd.	168
A.1f	A listing of output from the memory-image builder 'MBILD' for the example application of Chapter 6 ...contd.	169

LIST OF PLATES

PLATE		PAGE
5.1	A typical interchange between SBILD and a DISCO user via the HP 2648 A terminal	93

CHAPTER 1

INTRODUCTION

1.1 Introduction to the Project:

While computer control of chemical processes has been a major teaching and research interest in the department for many years, the emphasis has usually been on the control techniques rather than the software and hardware required to implement them. The principal tool for all of the work in this area since 1967 has been the IBM 1800 in the department's DACS (Data Acquisition Control and Simulation) centre. A number of projects which have used the IBM 1800 as a means for evaluation of advanced control schemes are described in Fisher and Seborg [23] and in a guide to the DACS centre [12] .

The replacement of the '1800' by a system of HP1000 computers, microcomputers, and dedicated intelligent devices, which is currently in progress, will require a major software development effort. Since Hewlett Packard does not supply process control software, a digital control package has to be written in order to maintain an effective service to the many research projects in this area.

Even a quite superficial investigation of this field would indicate, however, that a far greater contribution than just maintaining in-house service can be made. The rate of progress in computer hardware technology is forever widening the gap between modern hardware potential and current software capability. This phenomenon is analysed by Gordon and Spencer [27] in terms of the high 'man-time' cost of programming a new device. The fuller utilisation of state-of-the-art hardware for process control services has been a major objective in the DISCO project.

Schoeffler and Keyes [49] address the problem of identifying properties of computer languages and operating systems which result in qualities of efficiency and machine independence. It follows that software which simultaneously possesses both these characteristics should be easily implemented on a prototype hardware product and should give good performance. This reference and several others (Diehl [17], Lalive d'Epinay [38]) advocate modular sets of routines arranged in layers in order to achieve these goals. The highest level layers are totally application oriented while the lowest level are entirely hardware dependent. A new application may be supported by selecting 'vertically' the required layers and 'horizontally' the required modules. When this has been done, hopefully only a few gaps will have to be filled with custom-designed code.

The design of the software and data-base in DISCO is compatible with a representation in terms of hierarchial

layers of software modules. This means not only that portability of the system is good but also that modifications of a specific hardware configuration are facilitated. There is ample provision for the DISCO user to incorporate his code at a number of levels thus allowing the handling of problems of arbitrary complexity. It is intended that when implemented on a distributed computer network, DISCO should also be capable of handling sensor-based applications of arbitrary size. The ultimate flexibility is achieved through handling changes of operating environment by adding or subtracting software layers, changes in complexity by adding or subtracting software modules and changes in size by adding or subtracting processors.

The industrial user of currently available commercial systems usually has the benefit of a high-level, real-time language for special purpose programs. His custom routines cannot, however, be incorporated into a vendor's DDC (Direct Digital Control) system such as IBM 1800 DDC [16] . They have to perform their own synchronisation with programs with which they interact and they cannot take advantage of routines already in the DDC package. This means, for example, that extensive advanced control schemes which are not designed on a loop basis become difficult to implement. Adaptive schemes often need to be built using tricks which are not conducive to data-base integrity. It is considered that the provision of a standard interface for user-written software in DISCO is a significant step in overcoming these

difficulties.

1.2 DISCO in the Context of Other Work in the Field:

DISCO, from its inception, has not been carried out in the traditional style of university research projects. The work undertaken in this thesis has included comparitively little speculative investigation. The aims of the project are more concerned with development of an industrial process control system which will take advantage of new opportunities not yet implemented by commercial designers. These opportunities arise from hardware advances, modern software engineering theories and innovative design philosophies. It is intended that the complete DISCO system will be capable of handling applications comparable in size and complexity to those handled by commercial software suites. Industrial control systems in this category have been developed by:

(i) computer manufacturers such as IBM, Honeywell and General Electric

(ii) process control instrumentation vendors such as Foxboro, Fisher and Taylor

(iii) software consultants such as Biles and Associates of Houston and Schwartz and Associates of Edmonton.

The value of standardisation has been recognised by almost all users of process control instrumentation. Large petrochemical corporations with sophisticated centralised research facilities expend considerable resources on choosing a single flexible instrument which can be used for all their applications of a given type. This policy results in the instrumentation problem being solved once rather than many times for a number of applications. It means that personnel are easily transferred from one plant to another, a useful pool of knowledge from which new corporate users can benefit is created and bulk purchases can lead to considerable concessions on the part of the vendor.

Examples of this type of standardisation with respect to control computers are the widespread use of the IBM 1800 by Exxon, the use of HP 1000 computers by Hudson Bay Oil and Gas for pipeline control and the use of the PDP 11 family of computers by the CEGB (Central Electricity Generating Board) in England. Aspects of the latter project are described in Marsh [40] , Entwistle and Oldfield [22] , Wells [63] and Jervis [35] .

The need for standardisation of control computers and software in the process industries gives rise to intensive research and development work by commercial organisations. Cooperation between vendor and user corporations has produced software packages such as PROSPRO 1800 [3] and DDC 1800 [16] (IBM and Exxon) and complete systems such as TDC 2000 [62] (Honeywell and Exxon). The latter work is likely

to lead to the Honeywell product becoming an industry standard in the next few years.

TDC 2000 is, however, a product developed over a significant period of time and as such inevitably underuses technological advances which have taken place since its conception. Honeywell provides the control engineer with many of the advantages of distributed digital processing in terms of improved reliability and modularity, but in the author's view, the latest distributed technology has the potential for a far more flexible scheme. Table 1.1 summarises some advantageous and disadvantageous properties of the TDC 2000 plant control system.

Computer manufacturers are contributing to distributed intelligence in process instrumentation by developing network software, microprocessor-based service modules and high-level software tools for system development. The DEC PDP 11-03 is an excellent example of a product which incorporates these progressive design concepts. Software for the PDP 11-03 can be prepared on a compatible PDP 11 using 'big machine' development aids and down-loaded to the microcomputer for execution. True network program sharing, resource sharing and communication can be achieved by operating the DECNET package on such a system. The PDP 11-03 microcomputer can be interfaced directly to a process or it can operate instrumentation on an IEEE 488 [71] bus. Interfaces to CAMAC buses as defined in [72],[73],[74] and [75] are also available.

Table 1.1 Summary Critique of Honeywell TDC 2000 Properties

Advantageous Properties

Improved reliability by distributing control amoungst a number of hardware devices.

Easy expansion due to a modular structure.

Interfaces to a computer for supervision purposes.

Single coaxial cable as a bus medium.

Disadvantageous Properties

No user programming of the microcomputer unit.

Complicated bus protocol discourages interfacing of non-Honeywell components.

High cost for small applications because of the need for an expensive bus controller.

Communication packages such as Digital Equipment's DECNET and Hewlett Packard's DS/1000 are of considerable importance in the development of distributed process control systems. The important point to note about these software modules is that they handle all levels of protocol in program to program interchange in connected processors each of which is operating in a multi-programming environment. Moreover, both DECNET and DS/1000 operate with arbitrary network topologies permitting almost infinite flexibility in connecting processors. Both systems provide a high-level language interface which makes network communication no more difficult than access of files or I/O devices on a single machine.

The principal standards for the connection of process instruments to a computer are the HP-IB interface (IEEE 488 [71]) and the four CAMAC standards for parallel, byte-parallel and serial buses (IEEE 583 [72] , 595 [73] , 596 [74] and 683 [75]). CAMAC systems have achieved some acceptance for process control in the nuclear industry, but generally both buses have been designed for fairly small laboratory-type applications. The HP-IB has a particularly convenient user interface; it is language and computer independent and requires very minimal programming knowledge to operate. Severe problems are experienced in larger applications because of the need for time-consuming binary to ASCII conversions and the need for cooperation between programs accessing the bus.

In designing DISCO considerable attention has been given to the advantages and drawbacks of these various systems. The advantages of modularity and reliability demonstrated by TDC 2000 have been kept by supporting network operation in DISCO. The lack of flexibility in terms of user programming has been overcome by allowing custom-designed data-base records with custom-designed routines to operate on them. Furthermore, DISCO is entirely independent of the actual means of communication between processors and as a result can take advantage of improvements in network software and instrumentation buses as they occur.

1.3 Documentation of the Thesis Project:

Documentation of the project has been carried out at two levels. This thesis, at the highest level, describes the concepts involved in formulating objectives, carrying out the design and implementing a working version of DISCO. At the next lower level, user manuals discuss details of executing the programs in the DISCO suite[1] on the HP 1000 computer under the RTE IV operating system. These manuals also give information needed for software maintenance and upgrading. The minute details of the program code are explained in the lowest level of documentation, that is, in

---

[1] Suite: a set or library of related programs and subroutines.

comments within the program source code.

The thesis may be broadly divided into three sections. The first section, which includes Chapters 1, 2 and 3, specifies the scope of the work and defines objectives and requirements for the project undertaken. The remaining chapters deal with the design of DISCO, schemes devised to meet the objectives given the tools available and the first experimental implementation. Chapter 2, the project overview, has been written as a 'stand-alone' introduction to the DISCO system. As such it should be useful to the reader who wishes to have some understanding of the structure and capabilities of DISCO, but has no need for the detailed presentation offered by the complete thesis. A similar approach has been taken in writing section 5.2. It is intended that this section should also serve as a users' guide to data-base building facilities and that it should provide an introduction to the functions of each of the data-base utility programs.

Production of user manuals was carried out with extensive use of computer aids for text processing. The major manuals describing the programs BBILD and SBILD were formatted as disk files on the University's AMDAHL 470 V/6 computer using the software package PMT [44] . Other programs requiring less extensive introduction and explanation were documented by use of the software module PRDOC [4] on the department's HP 1000 computer. PRDOC interactively requests the user for a description of a list

of standard features and attributes of a computer program. It formats the user's reply to each question and stores the questions and answers on a disk file which may later be listed by anyone interested in the particular program. Table 1.2 and Fig. 1.1 are a guide to user manuals available for programs in the software suite.

Table 1.2 Bibliography of Documentation for DISCO Modules Implemented as Part of the Thesis Project

Program	Associated Operation	Location of Manual
BBILD	Structural definition of a data-base table by the table designer.	On disk at Computing Services, U of Alberta in FMT source form.
SBILD	Building of a data-base table by the DISCO user.	"
MBILD	Generation of a memory-image for a DISCO table.	Documented under PRDOC, stored on disk at Dept. Chem. Eng., U of Alberta.
LBILD	Loading of a DISCO table.	"
DISCO	AKT processing.	"
ACCES	Data-base fetch/store processor.	"
GPWRD	Data-base display/edit utility.	"
GPACT	Data-base display/edit utility.	"
GPSEG	Data-base display/edit utility.	"

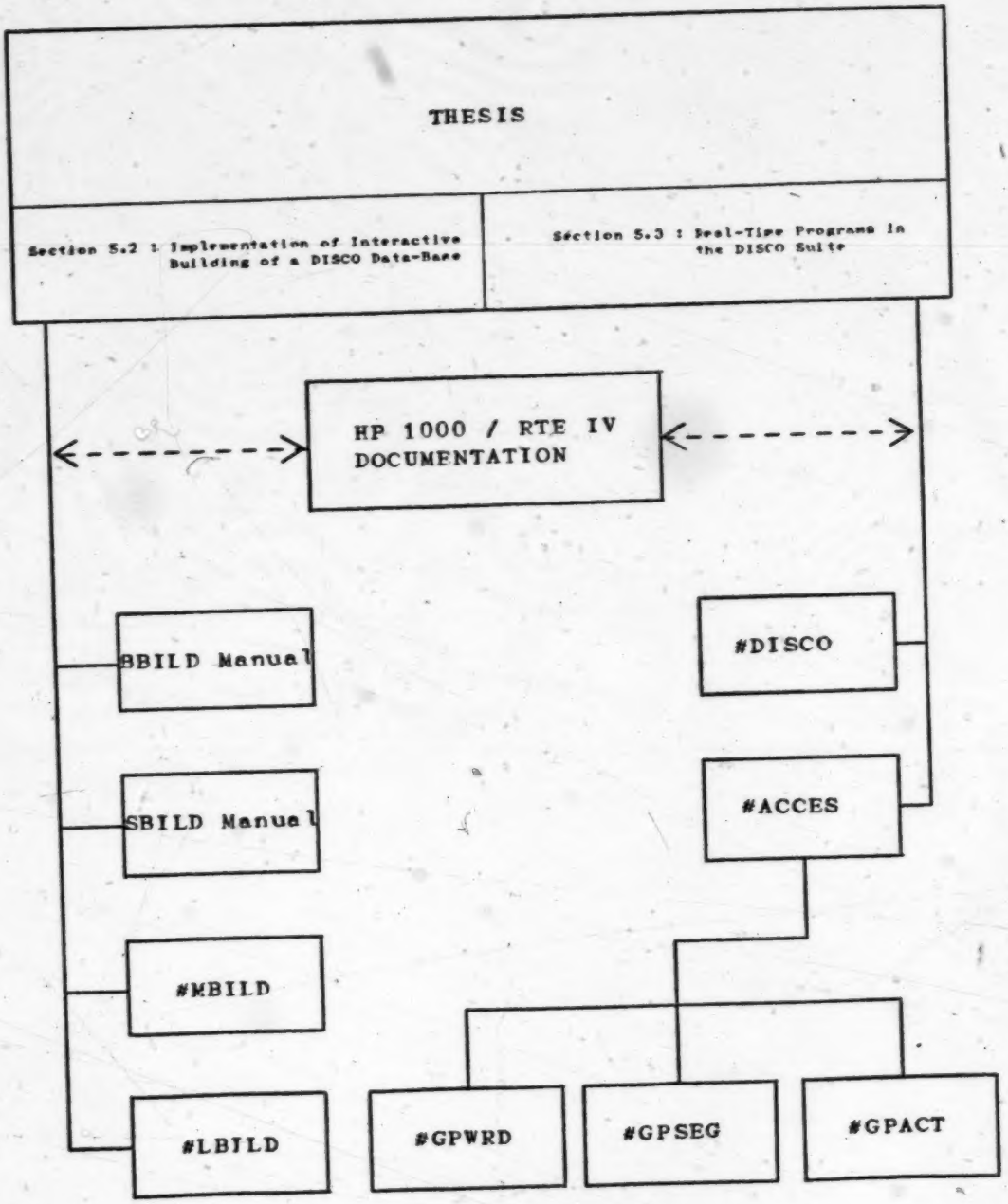


Fig. 1.1 Documentation of DISCO Modules Implemented as Part of the Thesis Project

CHAPTER 2

PROJECT OVERVIEW

2.1 Principal Research Interests:

The principal research interests in the DISCO project have been summarised below.

(i) Industrial process control: It is intended that DISCO will be developed to an acceptable industrial standard as a general computer control system. The incorporation of innovative design features in such a system without compromising industrial acceptability is one of the most important research interests in the project.

(ii) Advanced control capability: It is clearly a requirement for a computer control system such as DISCO that standard process control algorithms should be executed efficiently. These would typically include P+I+D feedback control, feedforward control and some non-linear control algorithms. It is intended that DISCO should

additionally handle any forseeable advanced control scheme. The advanced control category will include multivariable, adaptive and optimising supervisory schemes,

(iii) Distributed computer control: The DISCO project is concerned with investigation and application of distributed computer networks in a process environment.

(iv) High-level, real-time programming: The DISCO real-time software has been written in a high-level language. This has been facilitated by fast hardware speeds, an efficient compiler and micro-coded, high-level instructions.

2.2 The Process Activity and the Activity Record:

Since the advent of automatic control of process plants, control systems in the process industries have been described in terms of loops . This approach was clearly justified by the construction of the standard analog controller which almost always dealt with just one measurement and one output in a 'feedback loop' configuration.

Digital computer control systems are not, however, like the analog controller. They can handle not only single

loops, but also supervisory control, display functions, remote manual control and advanced control schemes. In particular, multivariable control and many types of adaptive control cannot be efficiently handled by a format which is strongly tied to a loop structure. Therefore, the DISCO system has been designed around a basic building block called an 'activity'.

A process activity will be defined for the purposes of this project as any service operation which the digital computer control system provides to the process. The operation is specified by an activity record which is part of the system data-base.

When designing an activity the user is advised to avoid any mechanisms which involve data-transfer between activity records. This, in effect, constitutes an objective that separate activities should be computationally independent.

By the use of an activity-based structure, DISCO has been made considerably more powerful and flexible than any loop-structured system. The control engineer can choose any combination of interacting process inputs, algorithms and process outputs to define his control scheme in an activity record.

The data-lists which specify components of the entire activity operation are known as segments. Each type of segment is processed by a separate routine in the control program. Each of these segment processor routines performs a

self-contained, software process service such as input acquisition, output transmission or alarm handling. The interface between the DISCO executive and the segment processors is identical for all segment types. In this way the addition of new segment types and their associated processors to the control system has been made very simple.

The organisation of a DISCO data-base in a large-scale, process control application such as a refinery is illustrated in Fig. 2.1. The figure shows how table-driven processing of input, control and output operations for a single feedback control scheme may be defined by segments in the data-base. The segments themselves are grouped in activity records which implement self-contained services such as those shown in the figure.

Since a segment processor may be designed to perform any function of which the computer hardware is capable, operations of very much greater complexity than those shown in Fig. 2.1 are possible. A selection of operations which may be defined by DISCO segments and executed by table-driven segment processors is presented in table 2.1.

The above description views DISCO favourably in comparison with DDC packages which are loop structured and dominated by a vendor-supplied program which provides no user entry points.

Clearly trade-offs have been made to achieve the desired flexibility. The variable format for the activity record implies that the data-items can no longer be located

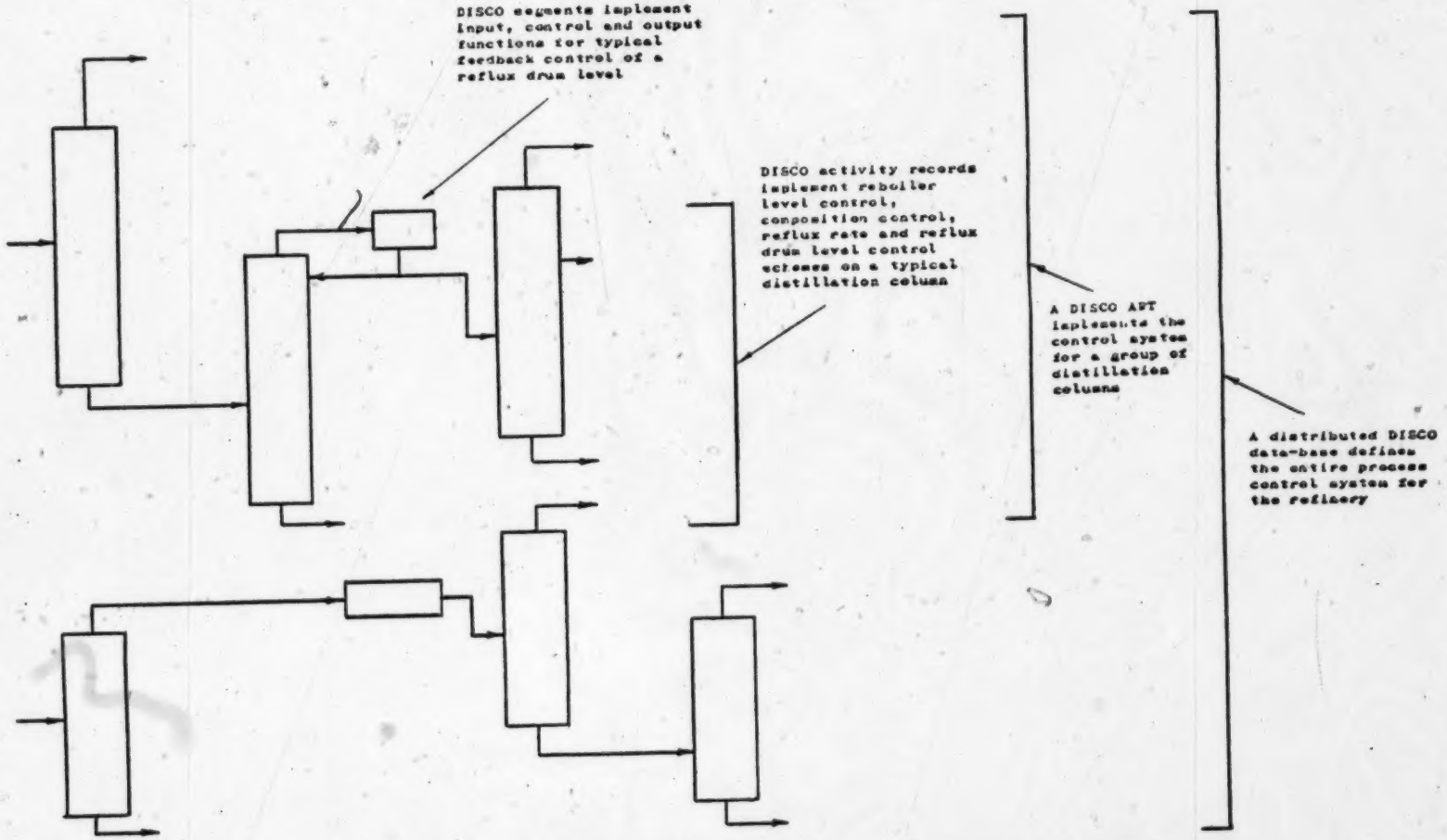

DISCO segments implement input, control and output functions for typical feedback control of a reflux drum level
DISCO activity records implement reboiler level control, composition control, reflux rate and reflux drum level control schemes on a typical distillation column
A DISCO APT implements the control system for a group of distillation columns
A distributed DISCO data-base defines the entire process control system for the refinery

Fig. 2.1 An Example of Possible Applications of the Main Subsets of a DISCO Data-Base in a Refinery Control Scheme

Table 2.1 A Representative Selection of Possible Segment Processors

(i) Input: One or more segment processors may acquire inputs from instrumentation buses, 'front-end' process interface microcomputers, network buffers or other activities.

(ii) Output: Output processors may be implemented to perform output operations complementary to the input operations listed in (i).

(iii) Control: Control segment processors may execute standard or advanced control algorithms. They may also schedule higher-level supervisory programs as a result of timing criteria or conditions detected during table-driven DDC processing.

(iv) Filter: Filter segment processors may carry out a variety of input and output processing operations. Since segments may be arranged in arbitrary order, the filter may also operate on several intermediate process variables during activity processing.

(v) Alarm: Alarm segment processors may check high and low limits and other logical criteria as required for specific applications. Programs may be scheduled or other activities enabled as a result of the checks made.

(iv) Process Operators' Console: Process operators' console segments may contain pointers to key variables which are frequently changed or displayed. Alternatively they may define the handling of special console buttons, lights or horn alarms.

(vi) Arithmetic: Arithmetic segment processors would be essentially similar to high-level language interpreters. They would facilitate arithmetic and logical operations on process variables during activity processing. This type of capability is of great value in the synthesis of decoupling, feedforward and other advanced control schemes.

Table 2.1 ...contd.

(vii)	Data Accumulation: A data accumulation segment may contain timing information and pointers which define periodic logging of key activity variables on a peripheral storage medium.
Note:	Preliminary versions of input, output and control segments have been implemented as part of the thesis project. A design for an arithmetic segment has also been completed. This work is described in Chapter 6 and Appendix B.

by convention, component operations must be explicitly identified and status data is often necessary for each component operation whereas previously it was only maintained for a complete loop. These characteristics give rise to some memory overhead in storing pointers, segment ID's and additional status words. The design of DISCO relies considerably on the hypothesis that memory and speed optimisation at a cost to functional capability of process software is no longer economically justifiable. Gordon and Spencer [27] support this view in an analysis of software development costs in engineering and scientific applications.

2.3 Process Services Provided by DISCO:

It is intended that in its final form DISCO will be a complete process management, monitoring and control scheme. Most of the components necessary for such a scheme have been specified as part of this project and a subset has been implemented to build a 'first version' working system.

Fig. 2.2 summarises the services which will be available in a complete DISCO package by showing schematically all the program modules which will operate on the process data-base. The programs are broadly divided into two categories according to their synchronisation properties. The first group, which effects the actual process services and interacts with the process operator is

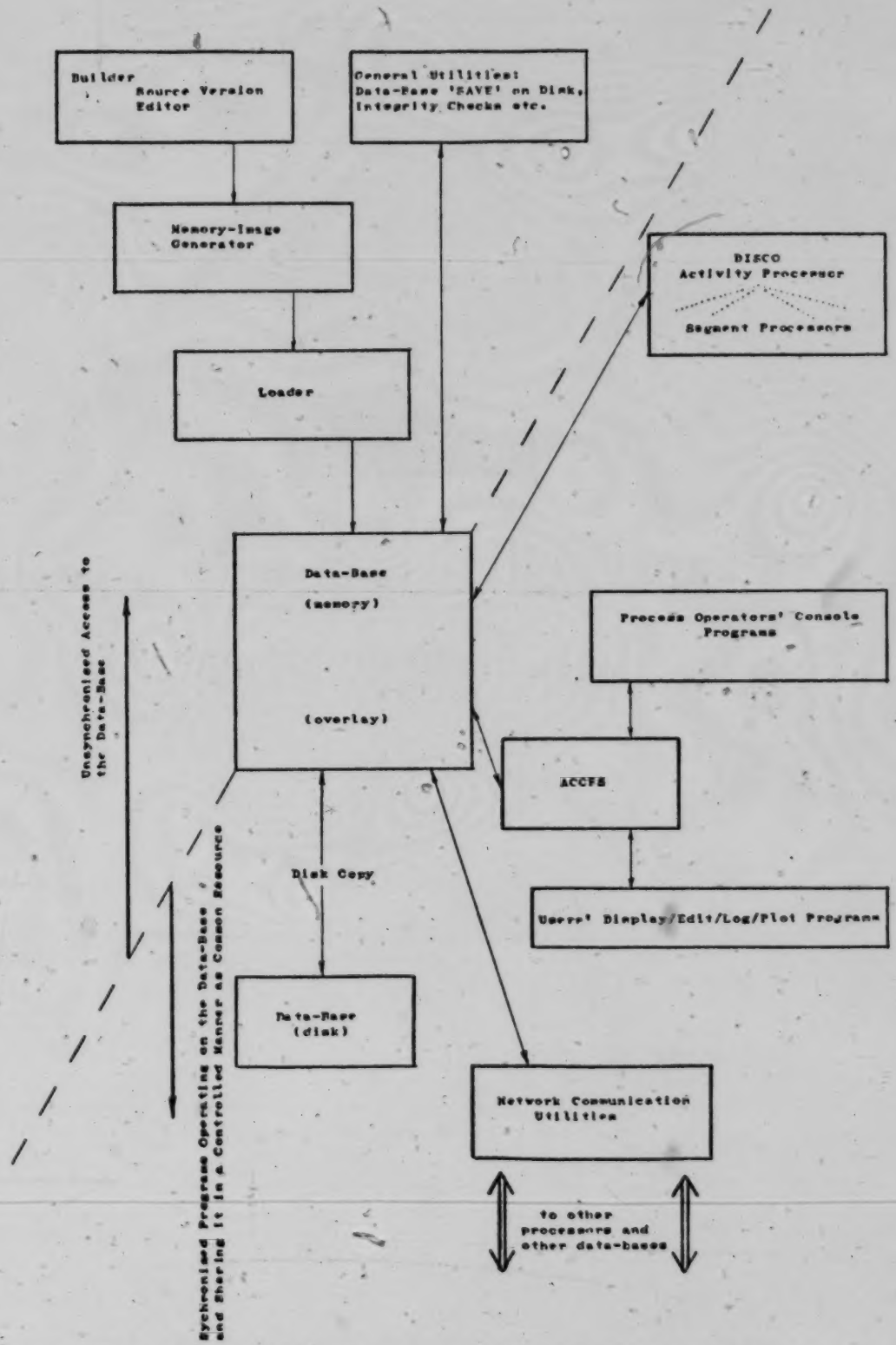


Fig. 2.2 Interaction of DISCO Program Modules with the Data-Base

synchronised within a DISCO poll to cooperate in sharing the data-base as a common resource. The programs in the second group are of a background utility nature. They are generally run at a much lower priority than the first group and are not real-time programs (as defined in Wirth [67]).

Fig. 2.3 shows how the general organisation of Fig. 2.2 may be extended for multiprocessor operation. Each processor node would execute its own set of the real-time programs. It is also likely that almost all nodes would have some support for interaction with the process operator. The background utility programs, such as the data-base builder, would however, be supported on only a very limited subset of processor nodes. In most forseen implementations this subset would be just a single node.

The following is an abridged description of functions carried out by each program.

(i) DISCO Executive: This program initiates activities and processes activity records when special flags or timing criteria dictate. Processing the activity records implements control and data-acqisition at the most direct level. For example, the program may carry out cascade control of liquid level in a process vessel as a result of processing an activity record.

(ii) ACCES: This program synchronises messages to activity

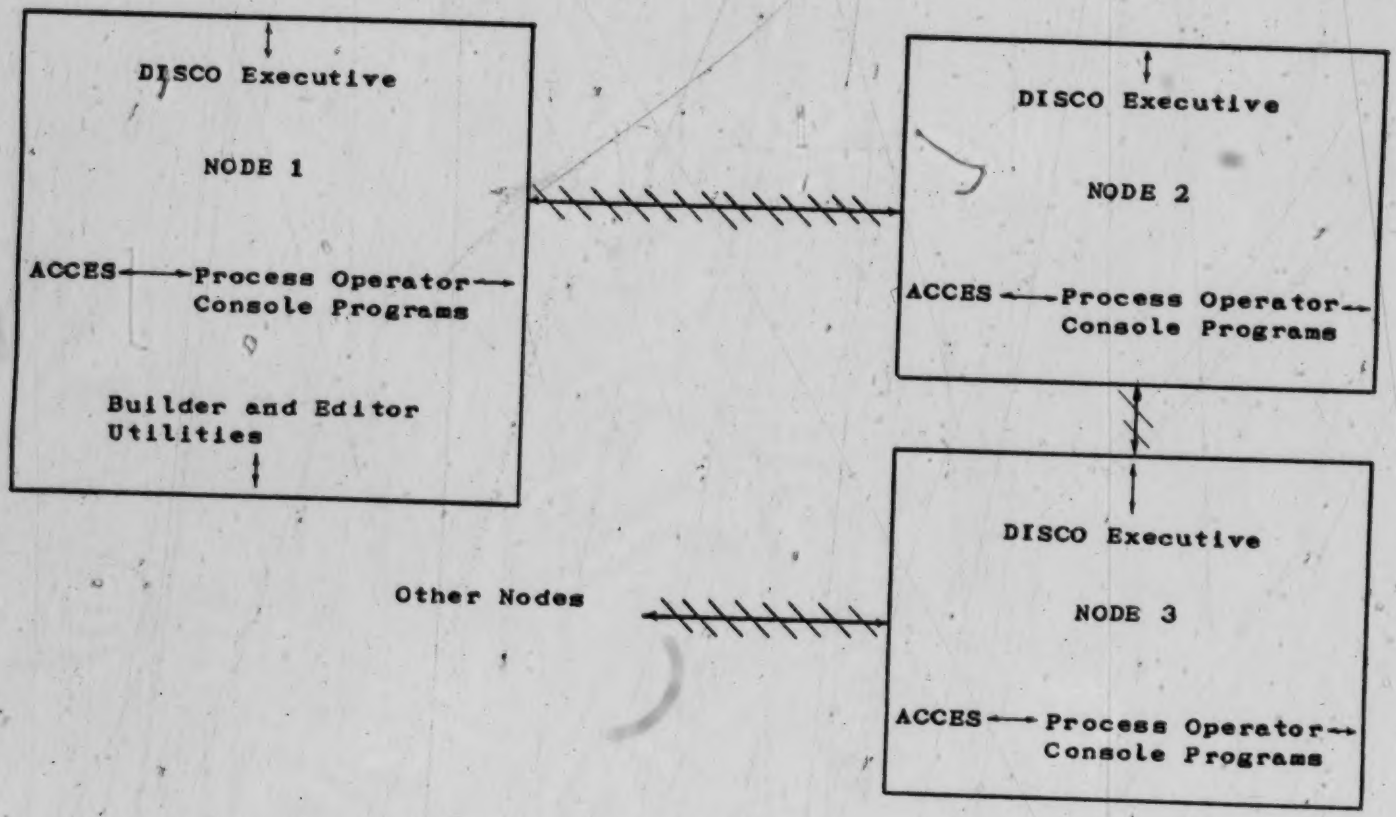


Fig. 2.3 Use of DISCO Program Modules in a Distributed Computer Network Implementation of DISCO

records in the data-base with activity processing carried out by DISCO. In its most commonly used mode, ACCES excludes reads, and updates to a specific activity while the activity is being processed. Conversely, ACCES also excludes the processing of an activity by DISCO while it is being updated.

(iii) Process Operator Console Programs: These programs will format displays and interpret requests for special purpose hardware consoles.

(iv) Users' Display/Edit/Log/Plot Programs: It is envisaged that this will grow to be a large set of special purpose programs. Some of these will format displays on computer terminals in a manner which might be especially convenient for a particular project, others may log values on disk for trend analysis and others still will probably be used to allow a general purpose computer terminal to be used as an operators' console.

(v) Network Communication Utilities: The node connections in a distributed network implementation of DISCO are made by the use of network communication utilities. This group of programs includes vendor's software such as DS 1000 and special

purpose software designed to interface DISCO to the network. The interface programs may make data from other nodes available for access during activity processing or they may acquire a complete ART which would be processed by the DISCO program in the same way as the local ART.

(vi) Disk Copy Program: This program will allow activity records for slow activities to be stored on peripheral, bulk storage devices such as a magnetic disk. Complete ART's on disk files will be copied into an overlay area and will be periodically processed by DISCO. The poll time would normally be greater than that for in-memory activities.

(vii) Build/Edit/Load Utilities: These utilities permit interactive building of activity records and editing of a source form of the data-base tables. The operational modules for these utilities have been written as part of this project but further work will be required to make them available from a more user-oriented executive.

CHAPTER 3

SCOPE AND CAPABILITIES OF DISCO

3.1 Problems Handled by DISCO:

DISCO has been designed to handle all the process services commonly required of a modern computer control system. These include data acquisition, telecontrol, telemetry, supervisory control and feedback DDC. Future expansion, for which interfaces and entry points already exist in DISCO software, may be carried out to include the following additional features.

(i) Management information services and optimisation schemes: These may be incorporated by adding 'higher-level' software layers to the DISCO system.

(ii) Input and output from additional devices with presently unknown protocols: New bus types and interfaces can be incorporated by adding driver modules in low-level software layers.

(iii) Batch and sequential operations: The DISCO data-base and data-base builder designs are sufficiently flexible to accommodate table-driven sequential control. This type of control would have to be governed by a sequential activity executive which would probably be separate from the DISCO executive implemented as part of the thesis project. Additionally, special segment processors would have to be written to handle the segments defining sequential operations.

3.2 Design Objectives:

In developing a design basis for DISCO the number of operational requirements multiplied rapidly as each new aspect of the system was considered. Many considerations dealt with relatively minor details of the design. Some decisions, such as those governing program documentation, were subjective and might not apply in a different working environment.

This section of the thesis attempts to select the key objectives which resulted in definition of the fundamental characteristics of the DISCO system. The objectives have been divided into several categories, each category relating to a particular aspect of DISCO operations.

The categories of advanced control and DDC include facilities for implementing any presently forseeable

experimental control scheme. These facilities are necessary in the Department of Chemical Engineering to support work on the development of multivariable and adaptive control techniques which would be acceptable to industry (Shah [51] , Johnstone [36]). The same features of DISCO which enable evaluation of experimental schemes may, however, be used for implementation of decoupling or constraint control techniques which have already gained widespread acceptance in industrial applications (Shinskey [54] , Wade [80]).

At a more basic level than the categorised objectives, three principles have set the theme for the DISCO design.

I. Flexibility: All aspects of software design are dominated by the desirability of making the system versatile and suitable for future modification.

II. User-oriented design: All aspects of the design and implementation have been strongly influenced by the desirability of making features of DISCO compatible with development and operation by the human user.

III. Industrial acceptability: The DISCO design has at all times been influenced and constrained by the intended viability of the end product as an industrial computer control system.

3.2.1 Objectives with Respect to Conventional DDC Capability:

Table 3.1 lists in detail the key objectives for DISCO operation as a conventional computer control system. The process services supplied by such systems are discussed in detail by Smith [58] . The objectives of table 3.1 will be interpreted in terms of a number of examples.

The first example is that of a simple feedback control loop on a process plant. Typically a level may be controlled by sensing with some kind of level transducer and setting a flow control valve accordingly (Fig. 3.1). All combinations of P+I+D algorithms had to be handled by DISCO and switching from manual control to automatic had to be straightforward. Simple incorporation of new algorithms was also a requirement. If, for example, the level loop described demanded a special non-linear scheme then DISCO would have to accomodate it without major program revisions.

DISCO also had to be compatible with the use of a typical process operator's console with functions such as those shown in Fig. 3.2. Formatting, scheduling and processing of terminal-oriented displays could be done by a software layer at a higher level than DISCO. The only requirement here was a tidy interface between DISCO and such higher layers. Many console activities, however, require digital and analog process I/O. Hence, it was a requirement that such functions as response to operator push-button panels, mimic diagrams and computer-generated analog 'dial'

Table 3.1 Summary of Objectives for Conventional DDC Capability

1) The system had to support all Dept. of Chem. Eng. applications which were at the time driven by the IBM 1800. Performance and user interface had to be at least as good as '1800' capability.

2) The system had to give versatile support for a variety of data acquisition requirements. This support had to cover such operations as telemetry, monitoring and data-logging.

3) The system had to give versatile support for a variety of output services. These included telecontrol and output of predetermined sequences (reverse data accumulation).

4) The system had to support combination of inputs with outputs to permit loop-based feedback control. A variety of control algorithms had to be supported.

5) Switching from feedback control mode to input only (monitoring) or output only (manual control) had to be simple to effect.

6) A wide variety of input and output processing had to be supported. This included digital filtering of arbitrary order and simple arithmetic operations such as those required for ratio control or decoupling control.

7) DISCO had to permit interfacing of higher level software layers to carry out:

(i) logging
(ii) trend recording
(iii) plotting
(iv) console support

8) Accompanying utilities had to support building of data-base tables in an interactive, user-oriented manner. Modifications in the format of data-base tables were to require only minimal adjustments in the builder

Table 3.1 ...contd.

utililities.

9) Accompanying utilities had to support listing of the data-base in a user-oriented format.

10) Synchronous clock-initiated processing had to interface with asynchronous, interrupt-driven routines.

11) The data-base design had to be compatible with future implementation of table-driven batch and sequential control systems.

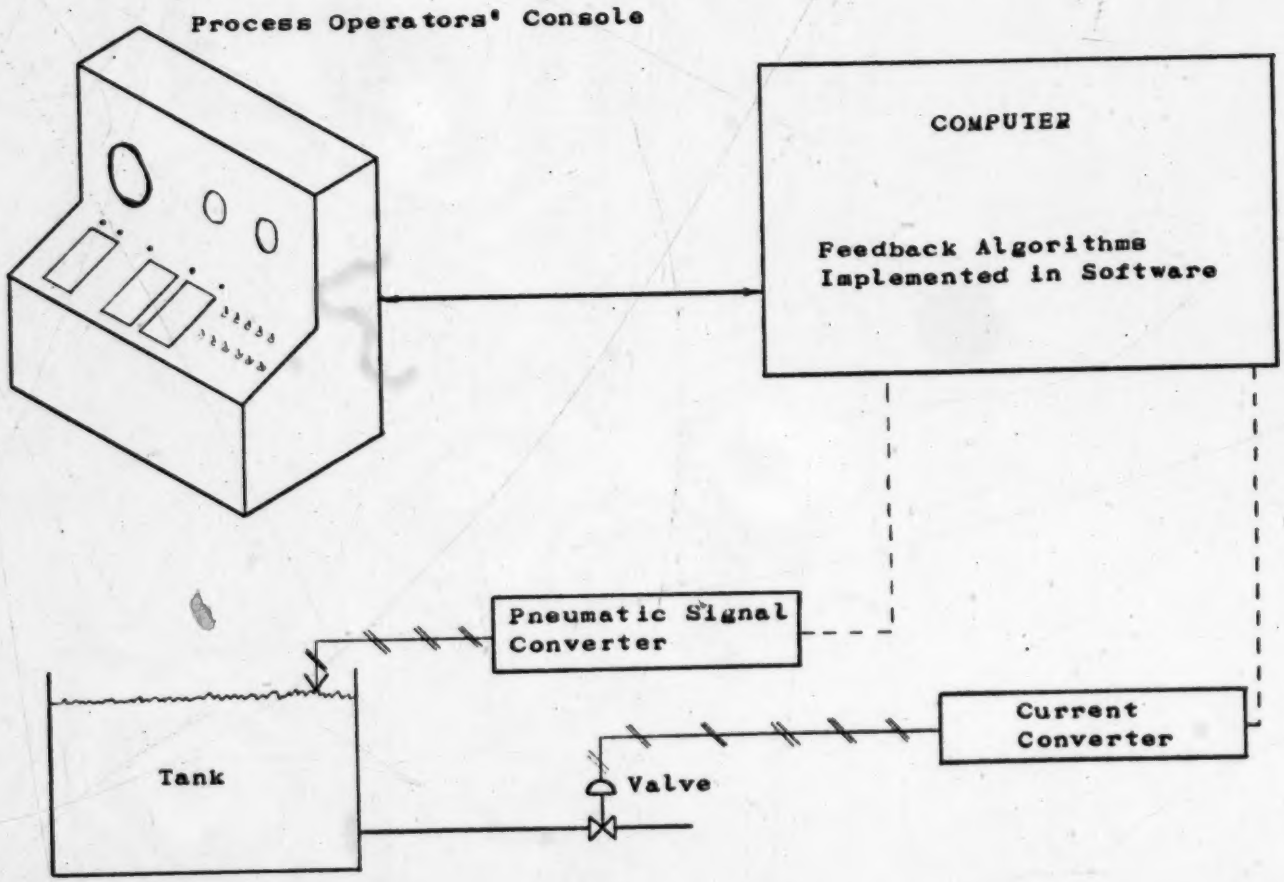


Fig. 3.1 Direct Digital Control of a Single-Variable Feedback Loop

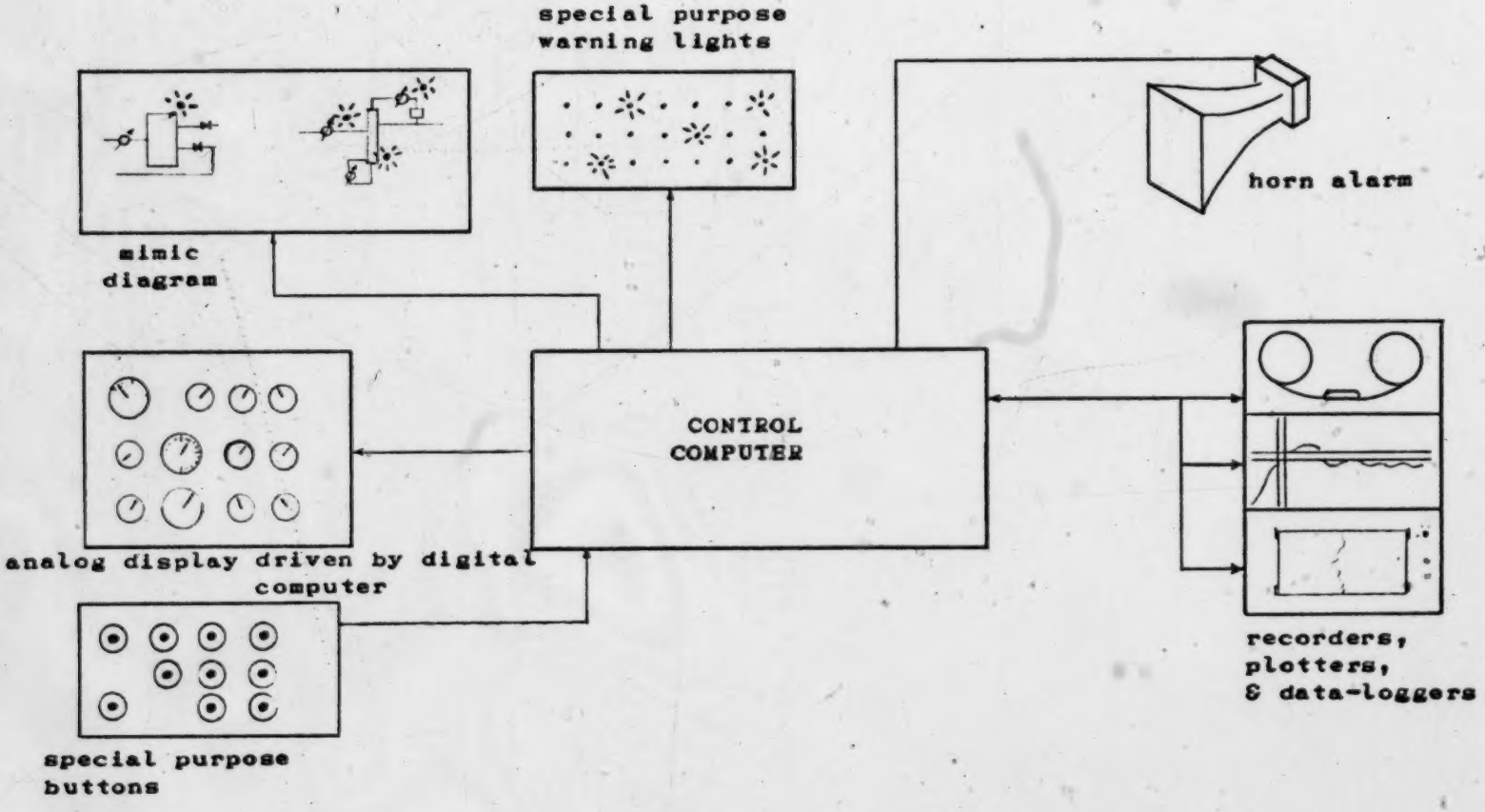


Fig. 3.2 Special Purpose Control Room Peripherals
Which Must Be Handled by Control Software

displays could all be handled by the DISCO software.

A third example of a DISCO application might be a computerised telemetry and telecontrol system. Such schemes are common in the power distribution and water distribution industries and have been described by Marsh [40] and Howard et al. [32] . A simplified illustration of such a scheme is shown in Fig. 3.3. The requirement here is for independent remote monitoring and remote control. Any feedback is supplied by the human operator according to loosely defined, qualitative rules which may be changed without access to the control system. The implication for DISCO was that memory and execution efficiency had to be good whether or not feedback was supplied by the control system.

The last example is that of laboratory automation. Consider an experiment in which an amplifier in a laboratory climatic cabinet is being evaluated for operation in temperature extremes (Fig. 3.4). The input and output from the amplifier may be controlled and monitored respectively by a power supply and digital voltmeter, both of which are connected to an IEEE 488 instrumentation bus. Thermocouples or resistance thermometers may be used to monitor the temperature in the cabinet and it is even possible that the temperature set-point may be supplied by the computer. A variety of experiments should be possible with such a configuration; DISCO would be responsible for supplying predetermined outputs to the power supply and cabinet while monitoring temperature and amplifier output. Report

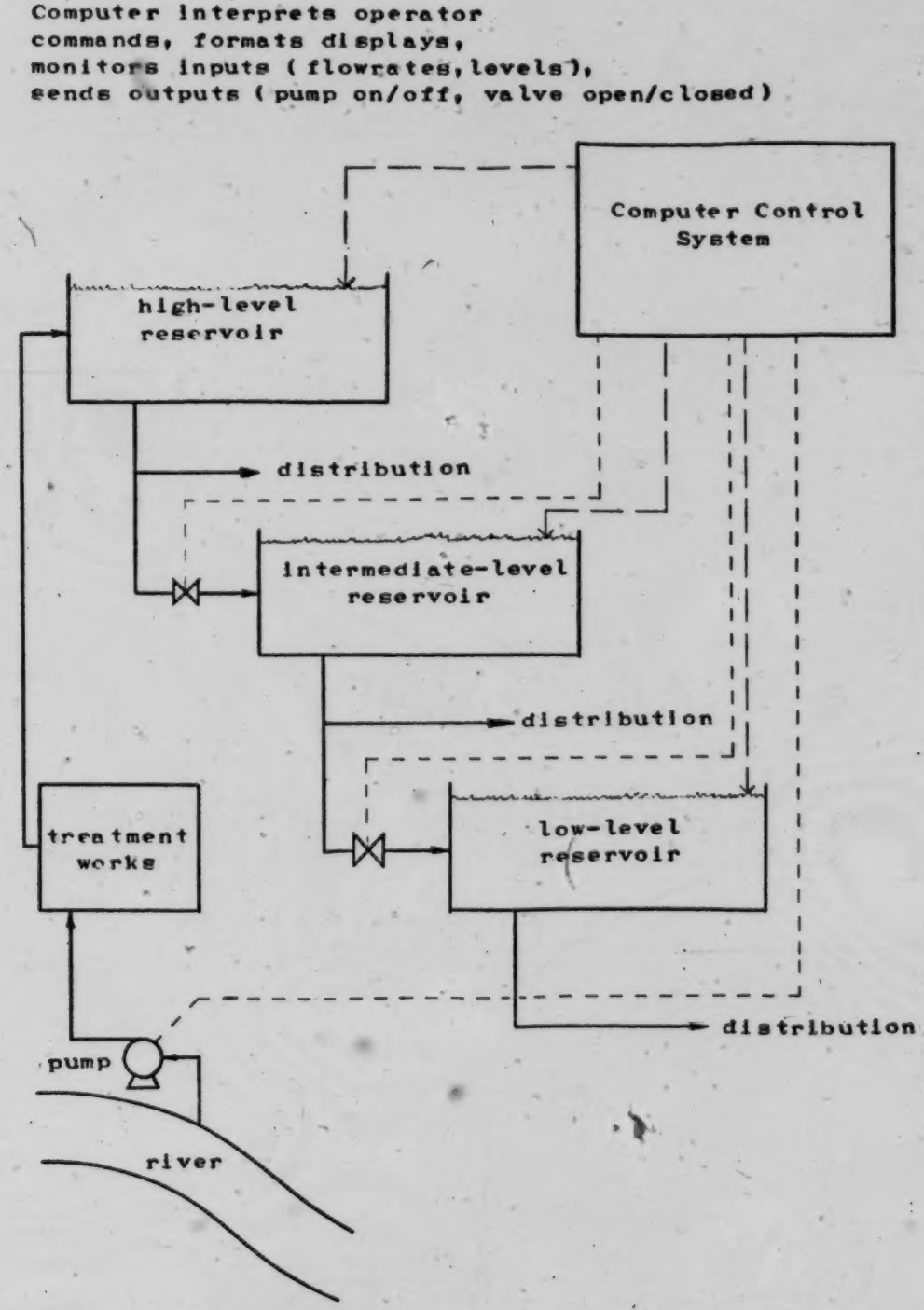


Fig. 3.3 Schematic Illustration of a Typical Application of Computerised Telemetry and Telecontrol in a Water Distribution System

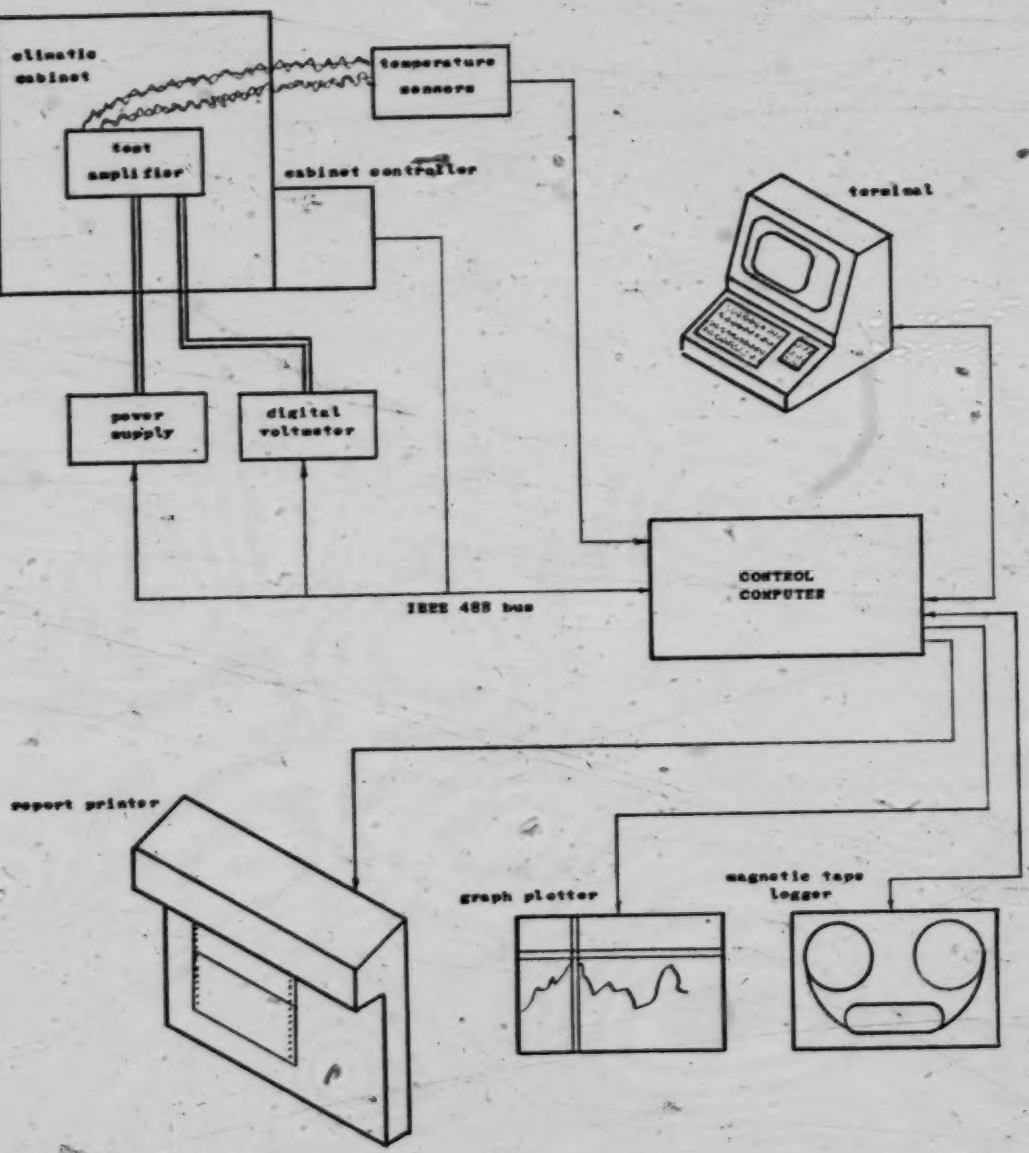


Fig. 3.4 Schemmatic Illustration of a Typical Laboratory Digital Control Application

generation, data storage and plotting could be implemented by software layers at a higher-level than DISCO. It was, however, a requirement that DISCO itself should efficiently handle all the real-time instrumentation support in such a control system.

3.2.2 Distributed Computer Network Operation:

A formal description of DISCO objectives with respect to operation on a distributed computer network is presented in table 3.2. The following discussion deals with some examples of possible schemes.

Fisher [24] describes the distributed network used for real-time applications at the University of Alberta DACS centre (Fig. 3.5). The network demonstrates a number of important features which have wide-ranging applicability and to which the DISCO design must be adapted.

At the highest level the local real-time system is connected by a data-link to a large, batch-oriented, data-processing computer. DISCO must be capable of acting as a means of data-collection for sophisticated analysis in a 'maxi' size computer such as the AMDAHL 470/V6 at the University of Alberta.

Within the Department of Chemical Engineering, three HP 1000 systems are connected by serial and serial-modem links. The computers also share two 50 MB disks via a common disk controller. DS/1000 communications software is used for

Table 3.2 Summary of Objectives for the Operation of DISCO on a Distributed Network of Computers

1) DISCO must be compatible with implementation on a multiprocessor distributed computer system. The roles of individual processors in such a scheme would be functionally distinct.

2) DISCO should be compatible with network connection of the processors or nodes in such a way as to permit cooperation and communication.

3) The topology of the network should be limited only by the communication software and hardware. DISCO itself must be compatible with any network configuration.

4) Resource allocation software must be incorporated either in real-time operating systems or in DISCO itself. This software must be capable of arbitrating between processors, at the same hierarchial level, which 'simultaneously' require the same resources.

5) DISCO operation on a distributed network must be supported by a distributed data-base. Jervis [35] describes how a centralised data-base degrades the benefits of distribution.

6) DISCO must be designed in such a way as to take advantage of the benefits of distribution which are provided by the particular software and hardware system on which it is implemented. Some examples of such benefits are:

   (i) Improved reliability by distribution of services among a number of hardware devices.
   (ii) Sharing of resources such as disks, line-printers and plotters.
   (iii) Easier maintenance, upgrading and expansion.

Table 3.2 ...contd.
(iv) Reduction in software complexity by use of dedicated processors.

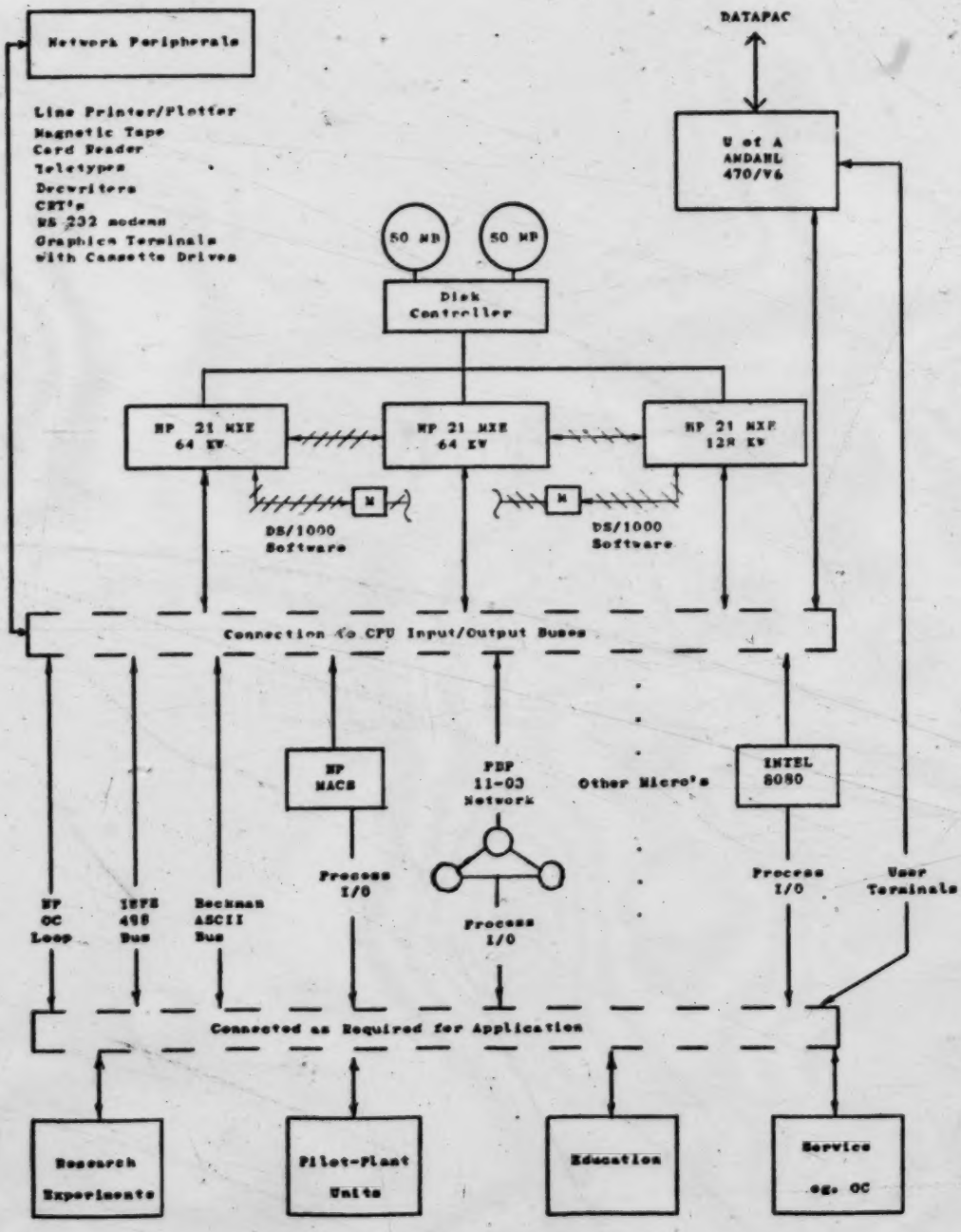


Fig. 3.5 Schemmatic Representation of the Distributed Network of Digital Computers Operated by the DACS Centre

communication, resource sharing and program loading and scheduling. DISCO systems in separate processors such as these HP 1000's had to be capable of cooperating and communicating to supply coordinated process services. The actual communication software was assumed to be in a layer below that of DISCO, thus making the control system independent of communication protocol at even the user-level. (A discussion of protocol levels including the 'user-level' may be found in Sloman et al. [55] .)

Goldsack [26] states that computational processes in the nodes of a network may be treated in the same way as cooperating sequential processes in a single machine. This means that synchronisation and deadlock problems may arise and can be handled by techniques such as those in Dijkstra [18] and Havender [31]. The deadlock problem arises when several nodes at essentially the same hierarchial level are all waiting for each other to release resources. Deadlocks may either be avoided by following specific rules (such as those given by Havender) or by detecting the deadlock and recovering from it. It was a requirement for DISCO that suitable resource arbitration should be available to facilitate shared use of network resources without prejudicing the integrity of the distributed system.

One of the most important features of a distributed computer network is the ease of expansion through addition of processors (nodes) and connections (arcs). This might occur in the system of Fig. 3.5 if an additional HP 21 MXE

processor was required for a new application. Conversely, one of the three existing processors might be removed for a special demonstration or for dedication to a new experiment. DISCO had to be structured in such a way as to facilitate addition or removal of processors in the network.

3.2.3 Objectives with Respect to Support of Advanced Control:

Table 3.3 lists in detail the principal objectives in providing support for advanced control in DISCO.

Fisher and Seborg [23] present a number of projects undertaken at the Department of Chemical Engineering which have evaluated application of recent developments in control engineering theory. Most of these were DDC applications in nature, but were implemented in supervisory programs because of the limitations of IBM 1800 DDC. It was an important objective in this project that such limitations should not be present in the new control system. The DISCO data-base and software structure had to provide flexible, comprehensive, table-driven support for multivariable control, adaptive control and other advanced DDC schemes.

3.3 Tools and Media of the Implementation:

The objectives or functional specifications for DISCO as defined at the start of the project have been stated and

Table 3.3 Summary of Objectives for Advanced Control Capability.

1) Multivariable direct digital control had to be easy to implement in DISCO.

2) It had to be possible to direct output from control calculations to any data-item in the data-base. This facilitated implementation of complex adaptive control schemes and made the control calculations independent of the output destination.

3) Supervisory control schemes of arbitrary complexity had to be allowed for.

4) The complexity of a feasible control scheme was to be limited only by the computing resources available. The structure of DISCO was not to be a limiting factor in determining feasibility.

discussed in section 3.2. The specifications leave a great deal of freedom with respect to the choice of computer systems, programming languages and process control instrumentation to be used in the implementation. DISCO could be implemented on a large data-processing machine connected to satellite, real-time, minicomputer systems or it could equally be implemented on a microcomputer network. This section describes the equipment and programming tools used in the thesis project to implement a DISCO system.

3.3.1 Computer System Hardware:

The computer system used for the implementation was the network of HP 1000 computers at the Department of Chemical Engineering. This system was used as an example in section 3.2.2 and is illustrated schematically in Fig. 3.5. The system has also been described by Fisher in [24] .

The installation consists of three HP 21 MXE processors running under the RTE IV real-time, multiprogramming operating system. A variety of standard peripherals are available including disks, plotting CRT's and a printer/plotter. The system supports a data-link to the university's AMDAHL 470/V6 computer, thus permiting use of more expensive devices such as multi-colour plotters, manuscript printers and an optical mark reader. Access to non-University of Alberta systems is available via the AMDAHL computer by use of the DATAPAC network.

3.3.2 Network Software:

A distributed computer network consists of arithmetic and logical programmable processors which communicate by means of some physical connection between them. The communication is facilitated by the adoption of a standard which defines electrical characteristics, mechanical characteristics and a protocol for the connection. The message-level and transmission-level protocols are usually closely related and together specify the control signals, timing signals and message format for the interchange.

There are two common techniques which are used for connection of communicating processors. These are listed below.

(i) Data-links: These use a communications protocol which is usually implemented in software. The link is terminated at interfaces which are similar to those normally used for a standard peripheral and the connection is usually serial.

(ii) Data-buses: The low level protocols for this type of connection are usually implemented in hardware. Commonly there is no interface as such because the bus may be an extension of an internal processor bus. At least control and timing information and frequently the data itself is transmitted on a parallel connection.

A distinction should be made between distributed processing in the context of the thesis and parallel execution of a single computational process. The sequential computational process is defined by Shaw [52] as:

> 'the activity resulting from the execution of a program with its data by a sequential processor'.

Spang [57] makes the point that the requirement in industrial real-time applications is for independent processing of functionally distinct, sequential, computational processes which together make up a task. DISCO has the capability to support this type of operation.

Intercommunication between the HP 1000 systems in the Department of Chemical Engineering is handled by Hewlett Packard's DS/1000 software. The package permits program-to-program communication, remote program loading, remote program scheduling and access to remote files and peripherals. The physical connection is made by serial and serial-modem links which, in the latter case especially, permit arbitrary spatial separation of the processors. The communications software has a user-level protocol which allows access from a number of high-level languages and makes the network topology transparent to the user. A minor disadvantage of DS/1000 is that routing through the network is not dynamic so that automatic substitution of redundant paths is not possible in case of a communication failure.

3.3.3 The Programming Language of the Implementation:

Recent improvements in performance of computer hardware have resulted in considerable interest in use of high-level languages for real-time programming. Whereas at one time all coding had to be done in assembler in order to satisfy real-time response requirements, the emphasis today is on maximising the efficiency of the programmer by providing sophisticated software development tools.

Schoeffler and Keyes [49] present convincing economic arguements to justify memory utilisation and program execution overheads associated with the use of high-level language programming. They discuss the respective merits of interpretive and compiler-based systems and present specifications for a hierarchial language processor which incorporates both techniques.

Williams [64] describes the joint ISA, IFIP and Purdue Laboratory for Applied Industrial Control project to develop a standard high-level language for real-time applications. This is a very large international project involving committees from the U.S., Europe, and Japan which are striving to achieve a definition of what is at present refered to as the 'Long Term Procedural Language' (LTPL). Williams presents thirty performance requirements for the language and eighteen necessary features of the implementation.

In choosing a language for the DISCO system there were several possibilities. It was undesirable to program in

assembler because of the longer development time, because program errors could affect other users in the multiprogramming environment and because very few of the people involved with DISCO had a strong background in assembler programming. Although there was interest in the installation of a modern, block-structured language such as PASCAL on the HP system, its use in the DISCO project would have created a number of problems. Considerable effort would have been involved in writing a compiler and designing interfaces to the operating system. Such interfaces would have had to be updated by the DACS centre when system updates were released by HP.

An important consideration was that excellent support for real-time operations is provided by the RTE IV operating system. Such functions as program-to-program communication, program timing and mutual exclusion of computational processes are available as subroutine calls to the real-time executive. Since these calls are supported in vendor-supplied languages there was considerable incentive to choose one of those already available.

With the above considerations in view FORTRAN IV was almost inevitably chosen for the DISCO implementation. In addition to the powerful operating system calls, standard ISA extensions [76] were accessible as a library package and a micro-coded fast-FORTRAN processor in the HP 21 MXE ensured efficient execution. The choice was considered to be consistent with industrial practice since FORTRAN is

available on a number of commercial computer control systems and it was also convenient because of the good familiarity of DACS personnel and students with the language.

3.3.4 Services Provided by the Real-Time Operating System:

Real-time and general management functions required by DISCO could be supplied by custom-written software or by a real-time executive. It has been particularly convenient in this project that the RTE IV operating system has provided almost all the necessary services of this type.

Diehl [17] presents the important features of a real-time operating system and those of a real-time programming language. Diehl's discussion on the division of duties between user programs and the executive are particularly pertinent in view of his affiliation with Hewlett Packard.

Table 3.4 shows features of the real-time operating system which have been useful in the implementation of DISCO. Had these features not been available it would have been necessary, in most cases, to develop software to provide them before work on DISCO itself commenced.

Table 3.4 Features of the Real-Time Operating System Which Have Been Used to Advantage in DISCO

Resource	RTE IV Management Functions
CPU	Program scheduling by operator request.
	Program scheduling by another program.
	Program scheduling by an external event.
Memory	Partitioning of memory with hardware mapping including some dynamic mapping.
	Management of buffers and communication areas.
	Program to program communication via the communication areas.
	Provision of reentrant libraries.
I/O Devices	I/O buffering.
	Queueing of requests.
	Facility for locking I/O devices.
	Concurrent I/O operations.
	Parity checking.
	Instrumentation bus handling.
Peripheral Storage	File management.
	Loading from peripheral storage.
	Background software development by use of editors, compilers, and loaders.

CHAPTER 4

DESIGN OF THE DATA-BASE

4.1 Table-Driven Processing:

In keeping with the dominating objective of flexibility in the project, DISCO has been designed in such a way that all operations of the control system are table-driven. While table-driven processing is a common feature of DDC systems, the principle has been extended in DISCO beyond conventional process control calculations. DISCO combines special-purpose, software modules with a flexible data-base structure to permit table-driven control of any operation that the computer hardware is capable of. Typical process services which may be implemented in this way are presented in table 4.1.

The advantages of table-driven operations in a multi-programming, real-time environment are very clear. If the code and operands of a control program were mixed in a single workspace then, for example, an independent display program would not be able to find the operands for its own purposes. Furthermore, passing of parameters between individual routines of the control program could in itself

Table 4.1 Typical Process Services Which May be Implemented by Table-Driven Software in DISCO
1) Polling of continuous processes and execution of control calculations. (Implemented as part of the thesis project.)
2) Asynchronous computation of control procedures for batch and sequential processes.
3) Interrupt processing.
4) Process I/O. (Implemented as part of the thesis project.)
5) I/O from/to local and network accessible storage media.

pose a problem. The very least limitation in such a case would be that all the operands and parameters defining process services would have to be contained within the workspace of the program. The mixed mode operation becomes downright impossible when one considers that alteration of parameters would require recompilation of the program itself. On the other hand, a separate, shared data-base allows access by a number of programs carrying out a variety of process services. Any program may be withdrawn from the system causing loss of only a subset of the available functions. Modularity is enhanced by allowing individual routines of a program independent access to the data-base with minimal internal passing of parameters.

The use of a separated data-base in a table-driven scheme also facilitates certain integrity measures. Identification of a 'dynamic' component of the control system definition assists protection of the system. The data-base can be backed-up on a secure storage medium, checkpointing of alterations can be implemented and the maintenance of an audit trail as described by Schoeffler [48] may be used to ensure that the 'dynamic' component can be reconstructed in case of failure. The fixed component, that is the program code, may always be reconstructed quite simply by reloading.

4.2 Choice of a Data-Base Structure:

Table 4.2 lists a selection of criteria presented by Dodd [20] , which affect the choice of a data-base structure. One of the most significant considerations in the design of the DISCO data-base has been the optimisation of fixed-period, cyclic processing of control calculations for a continuous process. The distinguishing characteristics of this type of processing are that most or all activities are polled on each periodic scan, retrieval and updating both need to be equally fast and efficient and keys are not used by the cyclic control program.

To deal with these requirements the data-base was organised as a sequential file of records which are processed, also in sequence, by the main control program. The scheme permits the use of directories to speed up access by other programs such as those used for process operator console support but, in general, it was considered that sequential searches would be adequate for operations of this type. Addition of records to the data-base is quite simple. However, insert/remove operations will necessarily be less efficient than they would be with random or list organisations. Such operations will be required only occasionally and therefore may be carried out by a 'copy and pack' utility.

Table 4.2 Criteria Which Affect Choice of a Structure in Data-Base Design
1) Required speed of retrieval.
2) Required speed of updating.
3) Typical number of records processed per access.
4) Number of keys used to access the data.
5) Frequency of insert/remove operations.
6) Similarity of records.

4.3 Compromises Made in Meeting the Design Objectives:

Table 4.3 shows the most significant data-base design decisions resulting from the design objectives of chapter 3. Items 1 and 2 of the table have already been discussed and justified in previous sections. This section will deal with the remaining points and attempt to present the reasons for structuring the data-base in the manner chosen.

Item 3 is related to the facility in DISCO which allows different process activities to perform widely differing roles. The storage of like data-items in a record could be seriously considered if a large number of identical operations were being undertaken by the control system. In such a case a convention would be possible whereby the i th data-item in each record was associated with a specific operation. DISCO precludes the use of a convention because an activity may use an arbitrary number of any type of operand or parameter. The only alternative to a convention would be the use of an absurdly large number of pointers and hence, the logical solution was considered to be the collection of all data relating to a particular activity within an activity record.

Segments of an activity record will be described in more detail in section 4.4. At this point it will be stated simply that they define some function, such as input or output, which is part of the process activity. Item 4 of the table states that internal segment formatting conventions may allow variable format and variable segment length. The

Table 4.3 Summary of Data-Base Design Features Chosen to Satisfy the Objectives of Chapter 3

1) The process activity , not the control loop, is the record basis.

2) The data-base is a sequential file.

3) A variety of types of data-item related to an activity are stored in an activity record. (As opposed to a record containing all setpoints, a second containing all inputs etc.)

4) Subsets of an activity record which define specific functions are known as segments. Segments are variable-length and contain no redundant fields.

5) Communication between segments is effected by:

(i) directed labels
(ii) a stack

Future development may additionally allow:

(iii) structured buffers
(iv) input/output operations on the data-base

6) Communication between activities will be allowed by the use of input/output operations on the data-base.

7) Any data representation required by a specific application is permitted. The following representations are recommended and have been used in the part-implementation carried out to date:

(i) on/off, single bit
(ii) positive integer, byte
(iii) signed integer, word
(iv) real data, 2-word
(v) strings and vectors, N-word

Table 4.3 ...contd.
8) The data-base design is consistent with processing by programs having a top-down, modular structure.

format and length in a specific application may be defined when the data-base is built by the choice of specific options. Any choice made by the user should, however, cause complete use of all space allocated as a result of his selection. The alternative to this policy would have been the inclusion of all possible fields for a given function in the segment. In such a case, selection of options would have merely indicated which fields were in use for a given application. The alternative approach was considered infeasible because of the large number of options which were desired for each aspect of DISCO's operation. The number of redundant fields and wasted space associated with unused options would have led to crippling storage inefficiency.

Items 5 and 6 of table 4.3 deal with the use during segment processing of results, parameters and operands from a segment other than that being processed. A typical example of the need for such an operation would be the accessing of an input as an operand in a control calculation. In this case the input acquisition and control calculation would usually be defined by different, separate segments. DISCO allows several means of reference of data-items beyond the segment boundary.

The first of these, the directed label, is simply a pointer to the data-item required in the remote segment. Work is currently continuing in the DACS centre to provide automatic generation of such pointers during activity record building. The second method, namely the stack, is simply a

'First-In-Last-Out' queue of integer data-items. Segment processors may perform 'PUSH' and 'POP' operations on the stack as the segments are processed in sequence. The method works well for typical feedback control operations in which operands, such as process inputs, can be used in calculations in the same order as that in which they are acquired.

Future work will include the implementation of an additional two communication schemes. The structured buffer would be a storage area in which specific data-items in a typical activity would have particular locations. For example, the program variable 'SETPT' could represent a location in which the current activity's setpoint was stored. The structured buffer could be passed from one segment processor to the next just as the stack can be passed. However, unlike the stack, the buffer would permit random access to variables within it.

The last data-base communication facility envisaged would be input/output operation on the data-base itself. For reasons of data-base integrity, any communication between activities will be restricted to use of this facility only. The technique will involve extension of ordinary I/O routines to operation on the activity records rather than I/O buffers. A compatible addressing scheme will be used to specify the ID of the activity record to be referenced and the required data-item within it.

Item 7 of table 4.3 recommends the use of data

representations which are easily handled by FORTRAN or by typical FORTRAN callable library routines. It is an important point, however, that DISCO imposes no limitations on data representations used in the data-base. Part-word fields, binary-coded decimal, Gray code representation and any other format which can be encoded in binary digits may be used in the data-base if appropriate processing modules are included in the DISCO software suite.

The data-base has been designed in such a way that subsets of decreasing order are processed by progressively lower-level software layers. This satisfies the statement in item 8 of table 4.3 for the entire data-base structure even to the individual data-item level. In the extreme case the data-item as the lowest order subset of the data-base is interpreted directly by hardware.

4.4 The Data-Base Structure:

The data-base in each processor involved in the digital control scheme contains:

(i) operands and parameters shared by synchronised programs running in that processor or in processors having network access to the table

(ii) operands and parameters stored berween subsequent runs of a single program.

4.4.1 The Activity Record:

The activity record contains data which is required for implementation of the process activity as described in section 2.2. DISCO recognises several 'types' or classes of activity in order to facilitate simple handling of commonly encountered demands. A typical example of use of a type designation would be the confinement of the operation of a loop-oriented, process operator's console to single-input, single-output activities. By adopting different type identifiers for multivariable and single variable activities a driving program for a console such as that described could discriminate between valid and invalid operations.

The control system also allows collection of activities into 'groups' to carry out tasks which are more conveniently executed by several cooperating activities. A typical example of a case where this might be desirable would be the implementation of cascade control where the slave loop requires a much faster sampling time than the master loop. In such a case two activities could be used to implement the two loops with a group designation being specified for the pair. Special properties of an activity group would be features such as use of directed labels for inter-activity communication, loading or deletion of the group as an integral unit and contiguous storage of the activity records comprising the group.

The format of the activity record header is shown in table 4.4. The header contains information such as the ID

Table 4.4 The Activity Header

Word	Description	Field-Length (bits)
1	Activity ID	16
2	Activity length	16
3	Activity enable/disable	1
	Local edit sync/unsync	1
	Global edit sync/unsync	1
	Clock initiated yes/no	1
	Data stack in use yes/no	1
	---- unused bits ----	3
	Processor network node ID	8
4	Number of segments	8
	Activity type	8
5	Poll count	8
	Sampling interval	8
6	Phase	8
	Group of activities ID	8
7	Activity input dimension	8
	Activity output dimension	8

number, length, status and timing information. The activity ID is network globally unique, it is always a positive integer, and its lowest order octal digit is always '1'. The length data-item effectively constitutes a pointer to the next activity record. Since there are no gaps between activity records and the logical order of records is the same as the physical order, the activity record table (ART) is, nevertheless, considered to have a sequential organisation as defined in Dodd [20] .

4.4.2 Subsets of the Activity Record:

The subsets of an activity record are shown schematically in Fig. 4.1. The record comprises an activity header and an arbitrary number of segments. In an activity record which defined the operation of a single-variable, feedback loop, separate segments would typically specify input acquisition, filtering, alarm processing, control and output transmission. The order of segments determines the order of performing the specified functions and more than one segment of a given type may appear in the same activity record. Since addition of new segment types and alteration of segment formats is extremely easy in DISCO, the functions performed by segment processors may always make full use of the capabilities of the available hardware.

Smaller subsets of the activity record may be defined as part of the convention for a given segment format. These

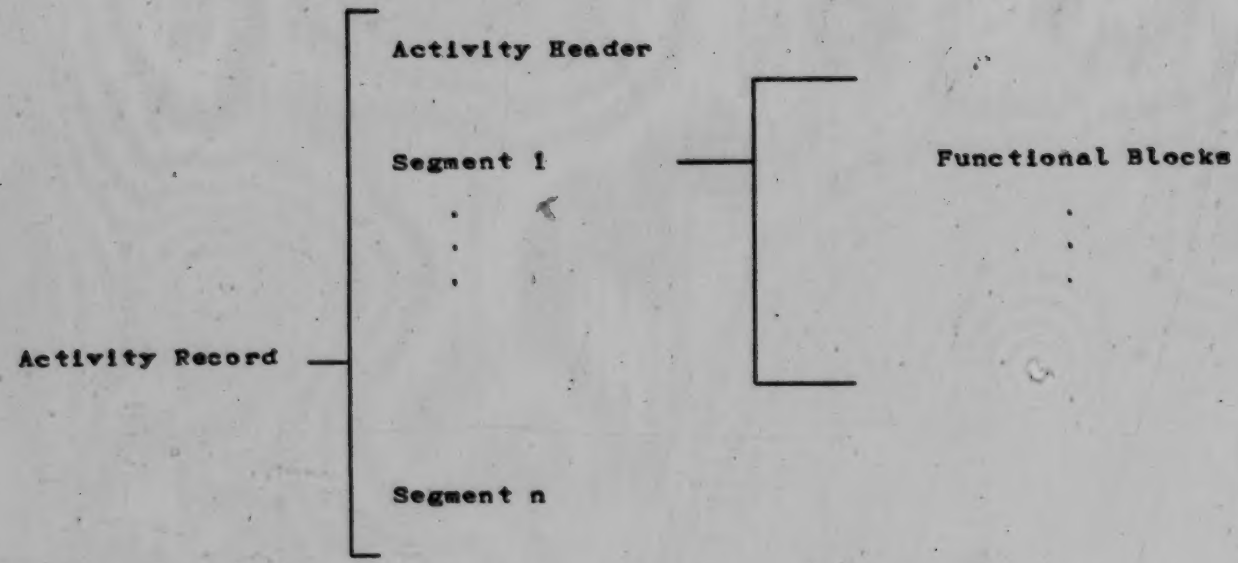


Fig. 4.1 Schemmatic Representation of Subsets of an Activity Record

subsets, known as functional blocks, may be used to provide a variety of formats or a multiple specification within a single segment. An example of this would be a segment which defined a number of filters in series. If each filter specification required a different format, functional blocks in a contiguous sequence could be used to achieve the multiple filter definition.

4.4.3 Organisation of the Data-Base:

Fig. 4.2 illustrates schematically the layout of the component tables which constitute the DISCO data-base. In the current implementation, the data-base is stored in the 'Real-Time Common' area of main memory in one of the department's HP 1000 systems.

Three important data areas exist apart from the ART described above. The first, the overlay ART area, is available for periodic superimposition of ART's from local or network accessible storage media. It is the responsibility of the program loading this area to synchronise with the ART processor and to obtain the overlay ART from the desired source location. The local ART processor scans this area after completing its operations on the local ART. One of the most important uses of the overlay area will be in processing of disk-resident activity records.

The I/O buffer area is the second major area outside

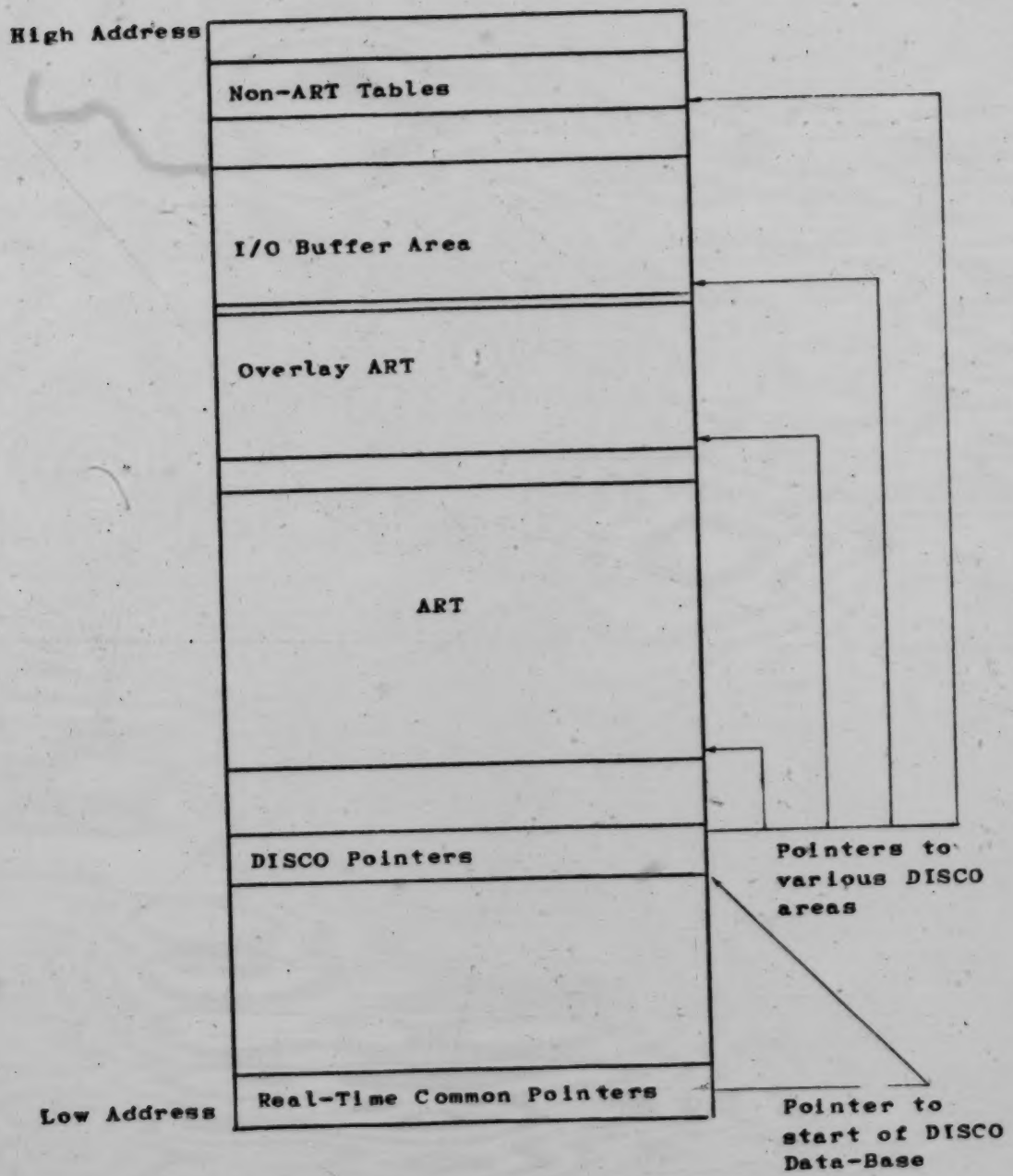


Fig. 4.2 Layout of the DISCO Data-Base

of the ART. This area will be used to store buffers for operands of DISCO input and output operations. The data records in this area will include buffers obtained from other network nodes as well as locally connected instrumentation. Fig. 4.3 shows how buffered and unbuffered I/O may be organised in a DISCO system. The buffers may be processed by:

(i) specially triggered DISCO scans which occur when new data arrives in the buffer

(ii) polling the buffer for arrival of new data as part of activity record processing

(iii) requesting the filling of a buffer, waiting and analysing the contents of the buffer on arrival, all as part of processing of a single segment (possible for fast I/O devices only)

(iv) asynchronously updating the buffer more often than it is processed.

The last major non-ART area is the general area shown in Fig. 4.2 at high COMMON addresses. This area contains tables referenced by the control system software from time to time, but not periodically scanned and processed like the ART. Typical examples of non-ART tables are given below.

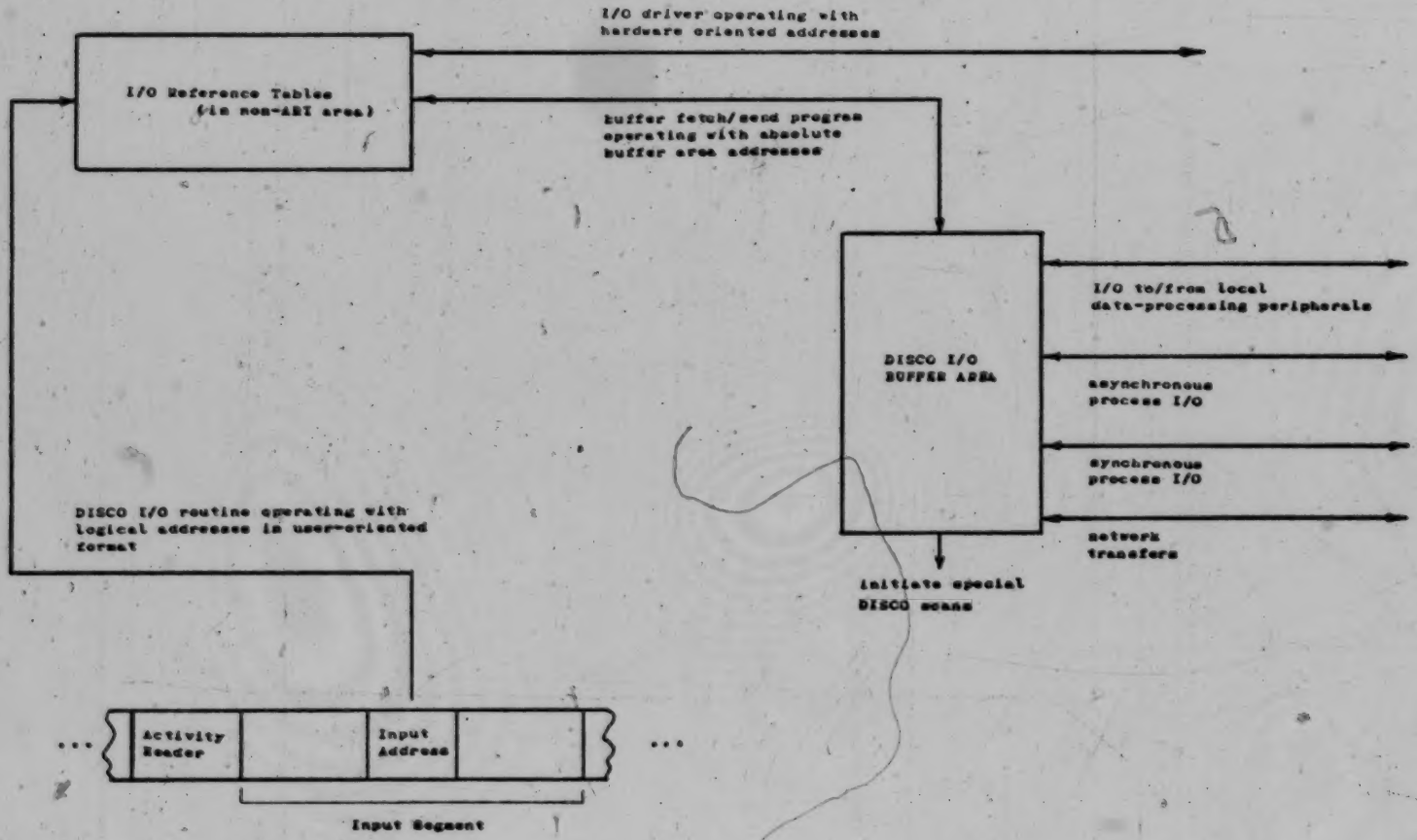


Fig. 4.3 Schemmatic Representation of DISCO Input/Output Operations

(i) I/O tables which provide hardware-oriented information required for instrumentation handling: These allow the user to specify an abbreviated logical reference to an instrument or part which would be used as a 'look up' key by DISCO when actually executing the I/O operation.

(ii) Engineering units tables: Zero and span values required for conversion of interface units to engineering units are commonly the same for a number of activities. Although these values may be included in input and output segments, memory efficiency can be improved by storing them in a non-ART table which may be referenced when required.

4.4.4 Distribution of the Data-Base:

A number of objectives have been formulated which relate to the processing of activity records by a distributed network of computers. These are a refinement and extension of the general distributed processing objectives presented in chapter 3. Limitations may be imposed by the characteristics of computer hardware or communications software in a particular implementation, but the structure of DISCO and the data-base are compatible with a full realisation. Objectives for distributed computer network

processing of activity records are presented below.

(i) Edit/fetch operations on the data-base in a remote node must be possible as part of ART processing. The addressing of remote data-bases in this manner must be implemented in a way which does not suspend real-time operations in the local node during the I/O wait period.

(ii) I/O operations using data-processing or process I/O peripherals attached to remote nodes must be possible as part of ART processing.

(iii) Processing of activity records obtained from a remote node must be possible.

(iv) Ideally the system should be capable of executing parts of a single activity on several network nodes.

The first three objectives may quite easily be achieved by fairly straightforward expansion of the present single-processor scheme. It is expected that most details of interprocessor communication will be successfully handled by Hewlett Packard's HP 1000 software. The fourth objective can be realised only when inter-segment communication is specified for the distributed case. Several obvious

extensions of the single processor techniques presented in section 4.3 may, however, be used for this purpose. These extensions are as follows.

(i) Directed labels could be used if all segments previously processed in other nodes were made available at the time of processing.

(ii) The stack could be used if it was passed from node to node as processing of the split activity progressed.

(iii) The structured buffer could be used if it was passed from node to node as processing of the split activity progressed.

4.5 A Comparison with Data-Base Design Procedures in a Different Working Environment:

An interesting comparison can be made between the design decisions made to meet the DISCO objectives and a somewhat similar exercise undertaken at the European Organisation for Nuclear Research. Daneels and Mead [14] describe the implementation of a distributed computer network to control a booster accelerator and a linear accelerator at CERN in Geneva. The project involved the design of computer control software from first principles

including the structuring of a real-time data-base. The result of this work was, however, considerably different from the result of the DISCO design exercise. This section will attempt to attribute the differences between DISCO and the CERN control system to specific characteristics of the respective working environments for which the schemes were designed.

Table 4.5 summarises the performance requirements of the control systems in the CERN project and in DISCO. It is immediately obvious that while the CERN system leaves scope for modification, upgrading and expansion, it does not claim to be a general purpose computer control package. The data-acquisition and control operations are uniform and well defined in the accelerator application, hence their specification was conveniently achieved by fixed-format, fixed-length records. All the instrumentation could be assumed to be accessible via a CAMAC bus, thus removing any need of providing other types of connection. The operator console support also did not require a sophisticated solution because the process consisted of large numbers of very similar mechnisms all of which could be handled by a similar format. There was no need for special-purpose consoles.

The accelerator control system, therefore, did not need the flexibility which DISCO provides by permitting a variety of single-variable and multivariable algorithms, logical referencing of instrumentation, control schemes of

Table 4.5 Performance Requirements of the CERN Control Scheme and of a General DISCO Scheme

CERN	DISCO
Several thousand process variables	Several hundred process variables
Many operator terminals	Few operator terminals
CAMAC instrumentation only	Several types of process interface
Many variables to be processed every 500ms	A few variables to be processed every second, most others on considerably longer scan
Control schemes limited to well defined single variable feedback	Control scheme complexity unlimited

Table 4.6 Design Features of the CERN Control Scheme and of a General DISCO Scheme

CERN	DISCO
Variables referenced by name and always located via directories	Variables referenced directly by location address
Hierarchial network configuration	Arbitrary topology
Fixed-format, fixed-length data record	Variable format, variable-length activity record with subsets

arbitrary structure and consoles of arbitrary structure. There were, however, stiff specifications to meet in terms of the number of variables handled, the speed of processing demanded and the number of operators accessing the data-base. These led to the use of a sophisticated data-base management scheme operating in real-time. The scheme supports the accessing of all data-base variables by name, the location being found from directories by parsing the name and using its components as keys. The speed requirements were met by limiting the application of any processor to a single, self-contained task. In effect, a scheme very similar to a DISCO system configured with one activity per processor.

The question arises whether DISCO could be applied to the control of the CERN accelerator. Since this is an application very much on the periphery of the field DISCO is designed to cover, the problem poses a real test for the flexibility of the control software. Table 4.5 reveals that the difficulties lie in the area of required speed of processing and data-base size. Improving the processing speed of DISCO for this specific application could be achieved by:

(i) adopting a fixed, feedback-loop structure with input and output segments which assumed a CAMAC interface

(ii) limiting the number of activities per processor node

(iii) use of assembler-coded segment processors.

Since DISCO interfaces easily with higher level software layers, a data-base management scheme such as that in the CERN design could be incorporated. The scheme would have to be capable of operating on records in the shared 'COMMON' area of main memory in which the data-base was stored. The ART processor or cyclic control program would still expect to reference process variables independently of any directories but if a fixed format activity were assumed, then a direct mapping between variable type and location would be possible. It is concluded that creation of a superset of DISCO by addition of software layers and streamlining of DISCO by selection of subsets of software modules within existing layers would make it feasible to control the CERN accelerator with a system such as that designed in this thesis project.

CHAPTER 5

EXPERIMENTAL IMPLEMENTATION AND EVALUATION OF DISCO

5.1 Introduction to the Implementation:

The component modules which make up a complete DISCO system have been described in section 2.3. This chapter describes how a partial-implementation has been carried out to construct a first-version working system. Further work, currently in progress in the Department, will refine and expand the existing DISCO software to a full realisation.

This thesis project will constitute a basis for the full realisation in the following way.

(i) The partial-implementation will provide a practical framework which already contains interfaces for most of the future additions.

(ii) Overall software design work carried out as part of the thesis project will mean that only structuring and coding of individual programs will be required for several of the outstanding software modules.

Tables 5.1 and 5.2 summarise the scope of this project in terms of the total work necessary to implement a DISCO control system.

A few areas of interest have been omitted from extensive consideration in the project. Design of operator console displays, for example, is of considerable importance in the industrial environment but has not been studied in detail here. In the university the 'operator' is usually also 'control engineer' and 'programmer', the applications are inherently safe and there are relatively few control variables. With these operating characteristics, sophisticated special purpose consoles are not justified and hence, their design has not been considered. Similarly, as has been stated in section 4.5, data-base management schemes for handling process variables have not been studied because in most computer control systems they are not required.

It is important to note, however, that considerable effort has been made to ensure that support for data-base access by higher order software layers exists even at the very earliest stages. The ACCES program which will be described later in this chapter provides safe, flexible and reliable processing of fetch/change operations on the data-base. These operations may originate from any program such as a console driver, data-base manager or supervisory program. ACCES provides a clean interface which ensures that development of such programs could be carried out without any limitations arising from the structure of DISCO software

Table 5.1 Features of the Part-Implementation of DISCO Undertaken in the Thesis Project
1) Implementation of the control system on a single processor only.
2) DISCO data-base resident in processor main memory only.
3) Three segment types implemented, namely those specifying input, output and control.
4) A limited number of algorithms available to segment processors.
5) Process I/O assumed to be on an IEEE 488 instrumentation bus.
6) Limited support for process operator interaction with the data-base.
7) Support for data-base building comprising most of the necessary software modules for this task.

Table 5.2 Summary of Main Areas of Future Work in Implementing a Complete DISCO System
1) Expansion to multiprocessor operation.
2) Development of support for process operators' consoles.
3) Development of a data-base operations executive.
4) Development of additional segment types.
5) Interfacing of applications.
6) Incorporation of data-base integrity and system recovery measures.

or the data-base.

5.1.1 Software Project Planning:

At an early stage in the DISCO project it was recognised that the implementation would become a large scale software development effort. This prompted a study of modern software organisational techniques. In particular, structured programming philosophies as discussed by Dijkstra [19], Yourdon [69], and Wirth [65] have been of great value in constructing reliable software mechanisms in DISCO programs.

The main advantage of the structured approach is that it facilitates verification of the expected program performance by inspection. This becomes possible as a result of the following features.

(i) Code for a complex program is reduced to a large number of very simple modules, each of which can be evaluated for correct logic at a glance.

(ii) The modules are constrained to interact by a single entry point and a single exit point in each.

(iii) The order of the modules in the program source listing is the same as the execution sequence.

Program clarity is further enhanced by well-considered documentation, human-oriented syntax and a tidy layout. These concepts are not part of the theory of structured programming but nevertheless assist considerably in making the code clear to the human reader. Before coding of any DISCO software was commenced, standards for the presentation of the FORTRAN source code were agreed with the DACS centre. These standards covered points such as:

(i) statements to be used in construction of logical mechanisms

(ii) use of labels

(iii) use of machine dependent features

(iv) layout and content of documentation.

A variety of error recovery and anti-bugging mechanisms have been used throughout the DISCO software. These ensure that any abnormal operation is detected and clearly notified to the user. In many cases the user is allowed to make adjustments which return the program to normal execution. It has been an objective that no abuse of any part of DISCO should ever result in the generation of error messages which cannot be trapped and acted upon by the DISCO software itself.

5.2 Implementation of Interactive Building of a DISCO Data-Base:

As a table-driven software system DISCO has its operation defined by a data-base. This means that each program in the system depends on common data tables, as well as its own fixed code, to determine logical flow and arithmetic operands at any time.

The tables contain DISCO activity records, segments, and hardware oriented information. Examples of a data-sequence which might form a segment and one which might form an I/O table are shown in Figs. 5.1a and 5.1b.

The purpose of the builder/editor/loader utilities is to build, load and maintain all the tables, such as those in Figs. 5.1a, 5.1b, which together form the DISCO data-base.

5.2.1 The Need for Software Support:

The simplest conceivable scheme for entering a data-base would be one in which the user constructed each table in memory-image form on some peripheral storage medium. All data values would have to be entered in a machine-compatible format (eg. octal) and where more than one field was stored in a single word, the user would have to assemble the fields into a whole word representation. Each of the many steps between an engineering problem description and a computer oriented task specification would have to be carried out manually. Due to the complexity of the data-base structure

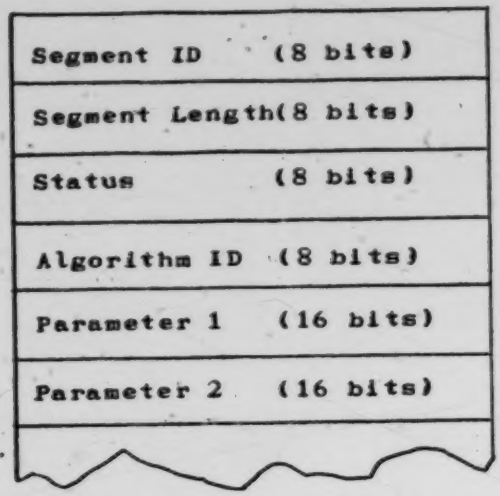


Fig. 5.1a
Schemmatic Representation of the Data Format of a Typical DISCO Segment

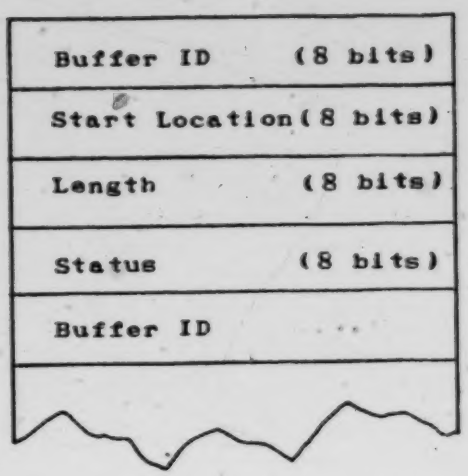


Fig. 5.1b
Schemmatic Representation of the Data Format of a Typical DISCO I/O Table.

needed to make DISCO suitable for a wide variety of applications, a manual approach to data-base building would be prohibitively error-prone and tedious.

At an early stage in the design of DISCO it was decided that a considerable amount of software development effort should be dedicated to automating the data-base building operation. The principal advantages to the user from the data-base operations software support are as follows.

(i) The software provides for task specification in terms of user oriented data representation (ie. decimal integer, decimal fraction or ASCII).

(ii) The software provides a user-oriented source version of the data-base from the user's input. This source version (the definition table) contains a description and the user's data-values for each field of the table.

(iii) The software facilitates activity specification by conversational prompting in a 'fill-in-the-blanks' format. This means that the builder program tells the user in 'plain English' which data-item values are required for the tables he needs to build.

(iv) The software provides for storage of data-table

specifications in a series of stages which represent the conversion from the user-oriented source input to the loaded list of binary machine operands. Considerable flexibility is achieved from the availability of these intermediate forms.

The advantages of this scheme are very similar to those accrued from relocatable object-code libraries which can be incorporated during loading of a program which has been compiled from high-level language source code. In like-manner, tables from a library can be added to those built by the DISCO user at any stage of the building process. The library tables do not have to undergo the entire translation process to which the user's input is subjected.

(v) The software provides for symbolic reference to data-item values during building. This means, for example, that the user could refer to a setpoint by the mnemonic 'SET' throughout the conversational interchange. The references are later resolved from symbol tables which can be generated or edited at any stage before the memory-image is produced.

(vi) The software provides for the use of labels for data-items that are pointers to other data-items in the

data-base. A typical example would be the identification of an output field in an output segment by the mnemonic 'OUT'. 'OUT' could then be used when building a control segment to point to the field in which the output should be placed. When the symbols were resolved the output mnemonic in the control segment would be replaced by the activity address of the output field, thus completing the implementation of the pointer.

5.2.2 Different Sources and Different Types of Information:

Individuals with a variety of skills and background need to be involved in data-base operations in order to make DISCO work. Representative demarcations are shown in Fig. 5.2 and the types of information supplied by each specialist are summarised below.

(i) Data Table Designer:

This would be a person whose main interest was software development of the real-time programs in DISCO. For example, he might be writing a segment processor or an I/O interface driver. If the software that he produced required a new type of segment or non-ART table then he would have to describe that table to the builder. This description (the 'working table') would be

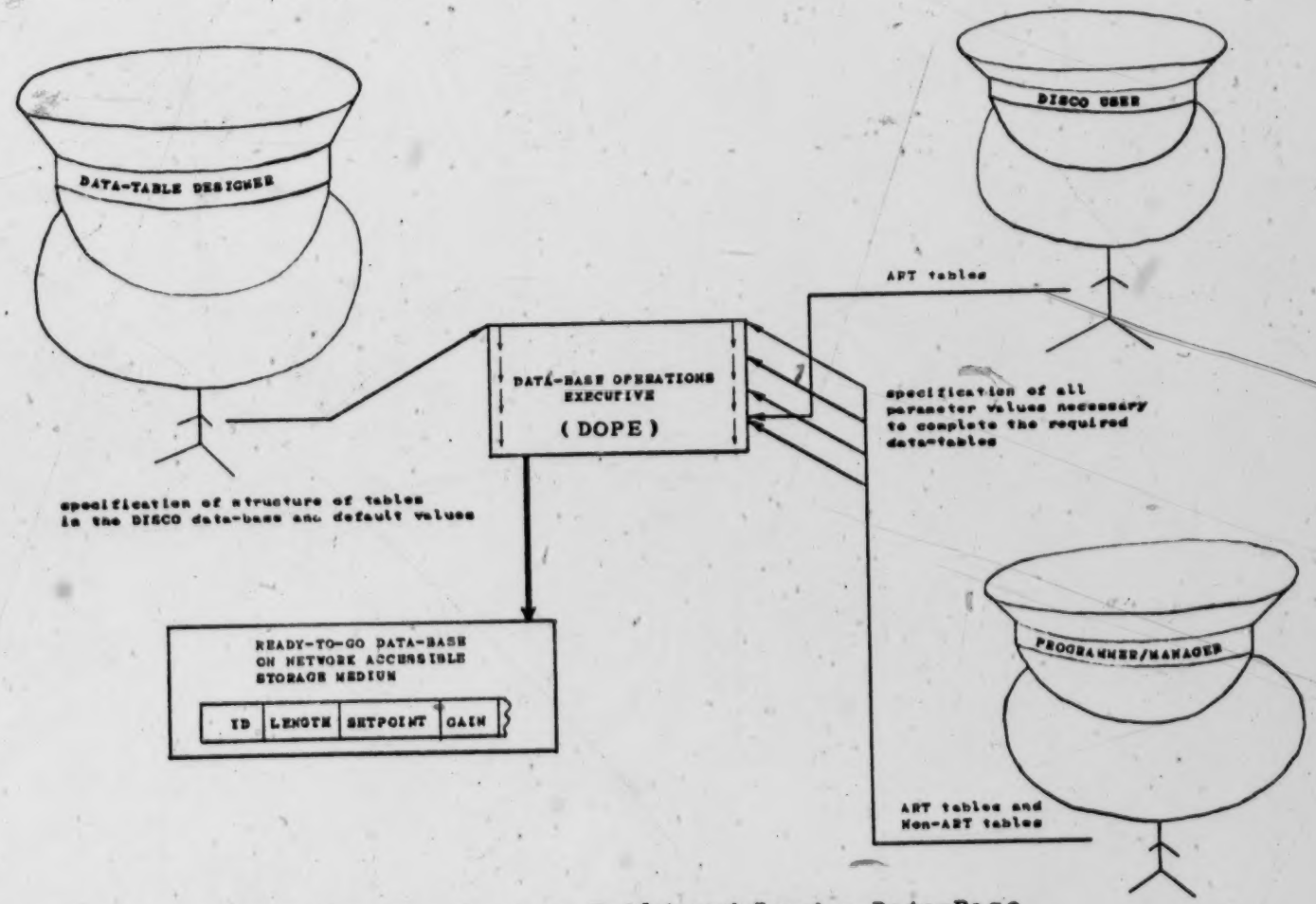


Fig. 5.2 Information Required to Build and Load a Data-Base

accessed whenever a user needed to build the new table to drive the designer's software.

(ii) DISCO User:

The DISCO user, in this context, is a person whose only interest is to use DISCO for a specific control or data acquisition application. The user will, therefore, take advantage of the conversational data entry facilities offered by the builder to supply control and data acquisition parameters to be incorporated in segments of the data-base.

(iii) Programmer/Manager:

The programmer/manager would be someone who has wide ranging responsibilities for the DISCO software. He may wish to construct library tables specifying activity records for commonly used applications but, in addition, he would have a more specialised role relating to non-ART tables.

The non-ART tables usually contain information which is required to interface with non-DISCO software (eg. operating system, network software) and to interface with hardware peripherals (eg. process interface, consoles). The tables may have to be edited when the operating system is regenerated, when hardware changes are

made, or when DISCO is updated.

5.2.3 Introduction to the Operation of Data-Base Building:

The documentation of data-base utilities in this thesis describes a complete design and partial implementation undertaken as part of the thesis project. The subset of facilities chosen for a first stage implementation represents a minimal scheme which was adequate to enable convenient and error-free data-base building.

A simplified illustration of the information required to construct a DISCO data-base is shown in Fig. 5.2. A more detailed overview is represented schematically by Fig. 5.3. This illustration shows the programs and data-tables used by DOPE (Data-base OPerations Executive) to provide the user with the required data-base utilities.

Even before a user attempts to build a data-base table, the table has to be designed and made to work with the routines which will be driven by it. At this stage the designer of the table and its associated software uses program BBILD to describe to the builder the structure of his new table. Once the builder has this information, it will be able to prompt future users for the data required to build copies of the designer's table for specific applications.

The structural information in the working table includes items such as bit-length of each table field,

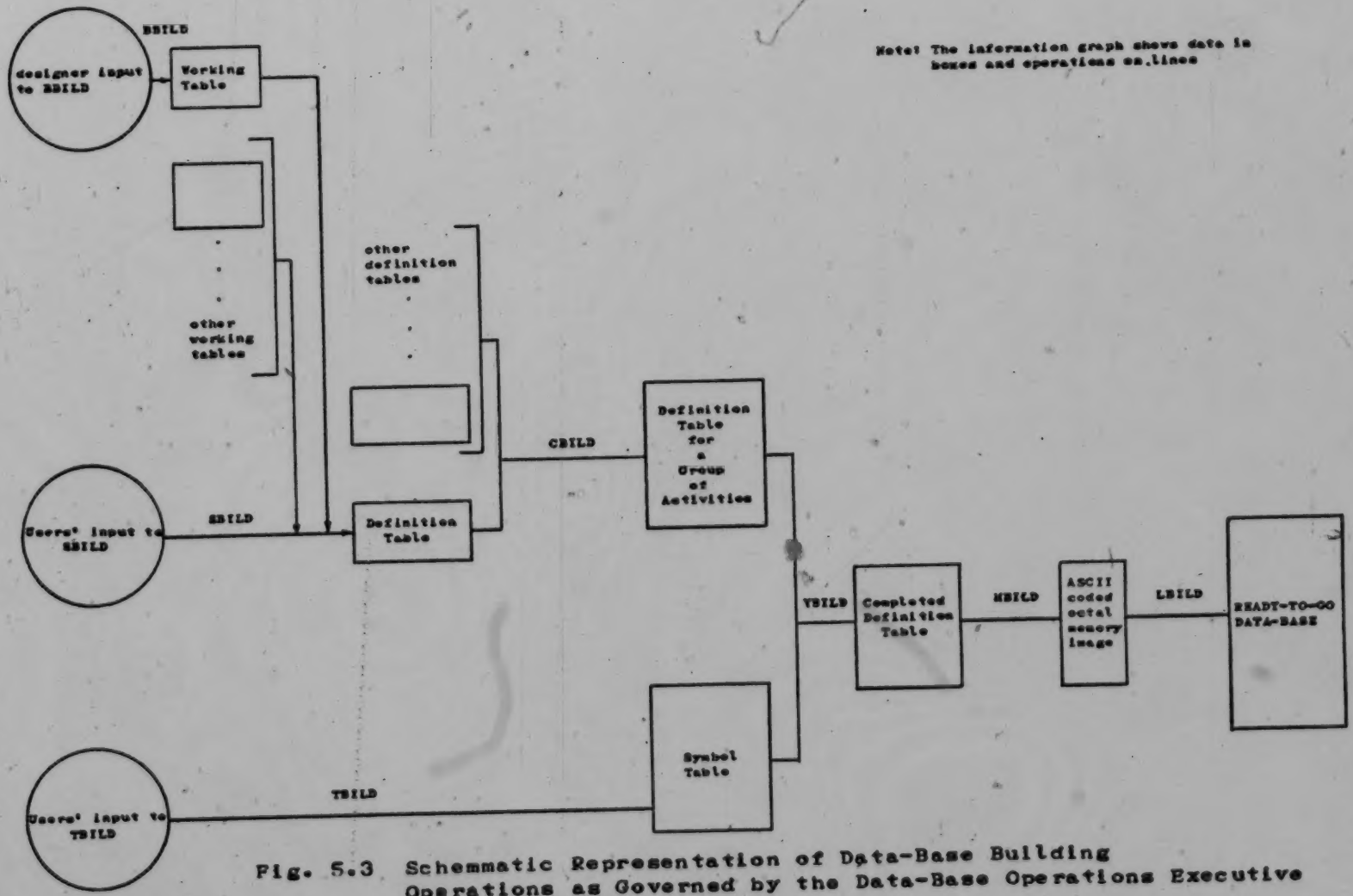


Fig. 5.3 Schemmatic Representation of Data-Base Building Operations as Governed by the Data-Base Operations Executive

default values, data-type (integer, real or ASCII) and format of variable-length vectors. Included also, is a description of each field in the table for use in interactive displays. This information is common to all the data-base tables of a particular type. Information required for a specific application such as actual data values, must be supplied by the user at build-time. This user input is solicited by SBILD during the construction of a definition table.

Plate 5.1 illustrates a typical example of interchange with SBILD via an HP 2648 A terminal. All information which may be changed by the user appears in inverse-video on the screen. After displaying such information the cursor is returned to the start of the inverse-video field to allow the user to perform visual editing. This system allows modification of the data-item descriptions as well as the data-item values in a particular definition table. For example, field 10 in the plate could be described as 'LEVEL SETPOINT' for a specific application simply by over-typing the existing text. The new description would appear on all future occasions when this field in this particular definition table was edited.

Each definition table has a 'one-to-one' correspondence with a data-base table being built for a specific application. The definition table not only contains all the structural information of the working table, but also data-values to be loaded into every data-item field.

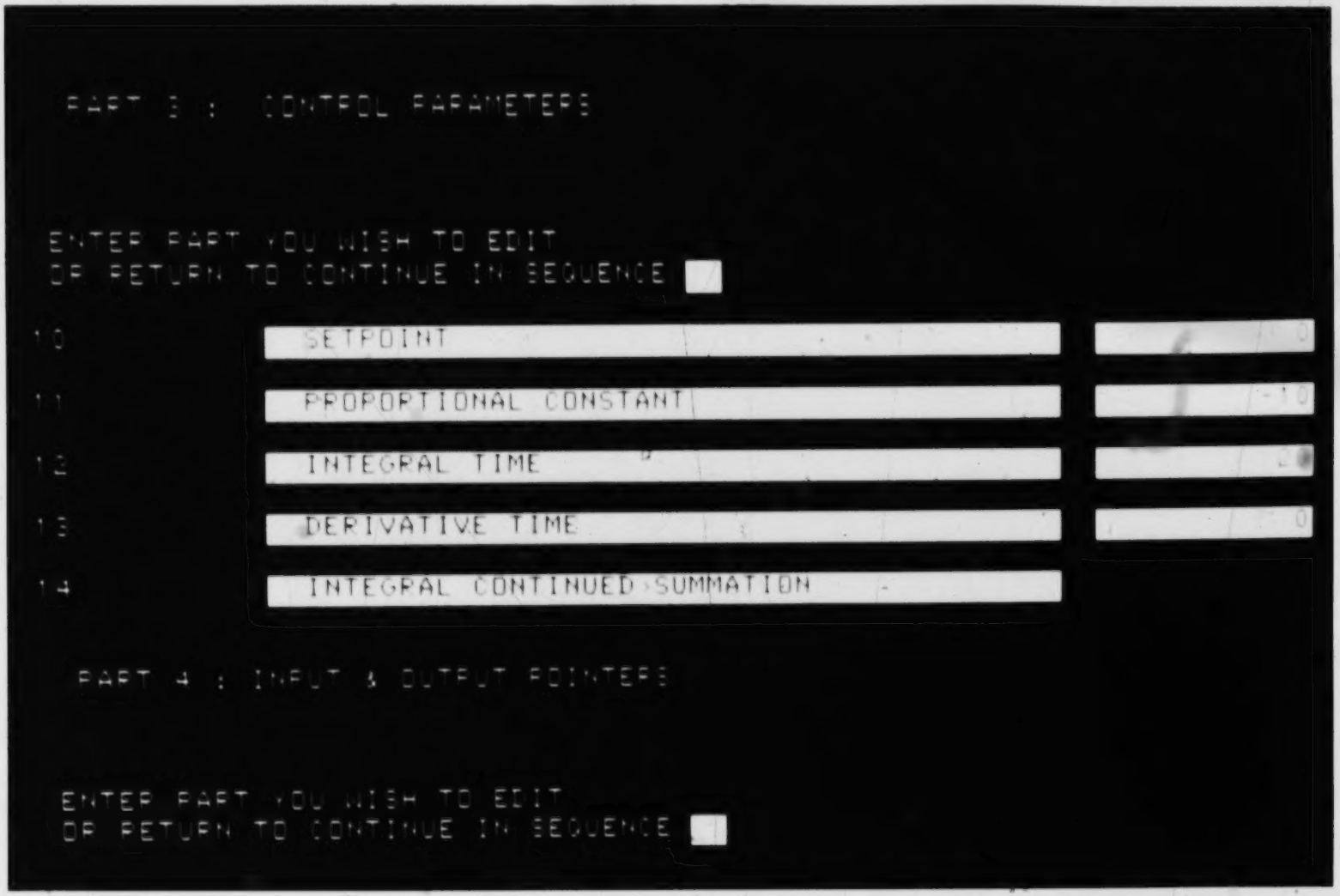

PART 3 : CONTROL PARAMETERS
ENTER PART YOU WISH TO EDIT
OR RETURN TO CONTINUE IN SEQUENCE
10
SETPOINT
11
PROPORTIONAL CONSTANT
12
INTEGRAL TIME
13
DERIVATIVE TIME
14
INTEGRAL CONTINUED SUMMATION
PART 4 : INPUT & OUTPUT POINTERS
ENTER PART YOU WISH TO EDIT
OR RETURN TO CONTINUE IN SEQUENCE

Plate 5.1 A Typical Interchange Between SBILD and a DISCO User Via the HP2648 A Terminal

The definition table is usually merged into a file of definition tables by program CBILD . When the file is complete, it forms a representation of a convenient subset of the data-base (eg. a group of activities). In parallel with these operations, symbols can be defined so that the symbol table eventually contains values for all symbolic references in the definition table file. At this point, the symbols and labels can be replaced by program YBILD, yielding a completed definition table.

The completed definition table is the last stage of processing for the user-oriented source version of the data-base tables. It is also the last representation of the data which can be edited by SBILD.

Once the completed definition table has been produced, a memory-image of the table can be generated by program MBILD. Since the image has to be manipulated on disk files, it is created in an ASCII-coded octal form. This makes it easier to examine the image and to include control records (such as an end-of-image marker).

The final operation required before the new table can be used by control and data acquisition programs is loading. The ASCII-coded octal memory-image must be converted to internal number form and loaded to the data-base area in the computer memory. When this is complete, the table is ready for use. Loading of data-base tables is carried out by program LBILD.

5.2.4 DOPE the Data-Base OPerations Executive:

The data-base building operations as illustrated schematically in Fig. 5.3 will, in the complete DISCO system, be governed by an executive. The design for this executive, which will be known as DOPE, has been carried out as part of the thesis project. The implementation of DOPE is currently in progress in the DACS centre.

When complete, DOPE will facilitate menu-driven, interactive selection and use of data-base utilities. The utilities themselves are completely interactive and hence the entire system will be highly user oriented. Fig. 5.4 shows the two-level menu structure supported by DOPE, additional lower levels are present in some cases where they are provided by the utilities themselves.

Table 5.3 summarises the services which DOPE will offer to the user of data-base building utilities. At present these services are carried out by the operating system, the file manager, the editor and manually by the user himself.

5.2.5 A Partial Implementation:

The programs described in section 5.2.3, together with DOPE, constitute the complete software support scheme for DISCO data-base operations. The partial implementation undertaken for the thesis project omitted programs which could be emulated by a combination of existing operating

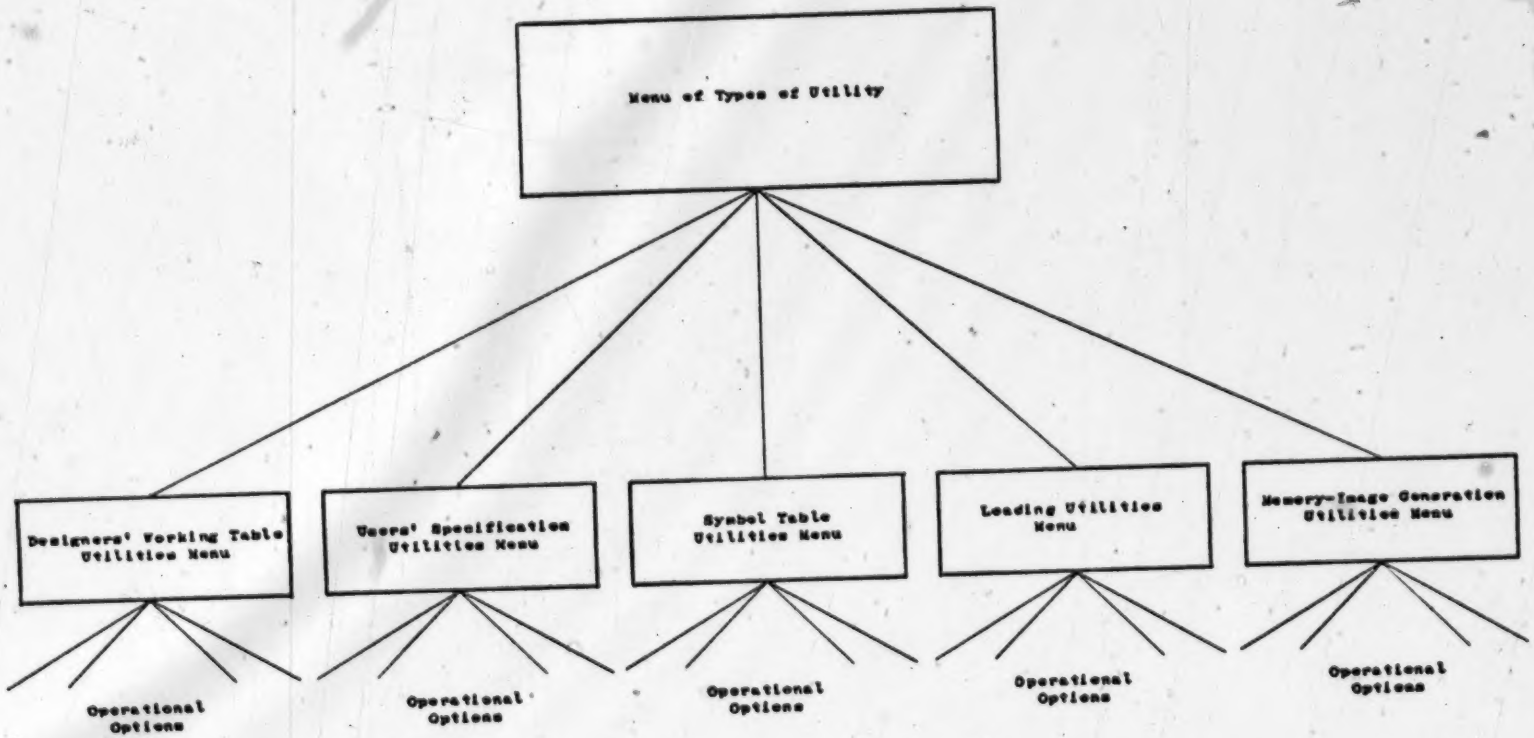


Fig. 5.4 Menu Structure of the Data-Base Operations Executive

Table 5.3 Services Provided by the Data-Base Operations Executive

1) Generation of menus for various aspects of data-base building operations.

2) Interpretation of user requests.

3) Recovery from meaningless or inconsistent requests.

4) Creation and deletion of files associated with data-base building operations.

5) Maintenance of directories of files associated with data-base building operations.

6) Generation of activity ID's for new activity records.

7) Maintenance of a load map.

Table 5.4 Interim Measures for Features of the Data-Base Building Utilities not Implemented as Part of the Thesis Project

PROGRAM	PRESENT INTERIM MEASURE
DOPE	The data-base operations executive is not available in the first implementation. All the data-base utility programs can be executed directly by commands to the operating system. Logic governing the sequence and conditions of execution of available programs is entirely the responsibility of the user.
TBILD & YBILD	Symbolic references may be supplied by the user when, for example, building a segment. If supplied they will be incorporated in the definition table. There is, however, no software support for replacing the symbols and labels. If used, they must be replaced by the editing facility in SBILD before a memory-image is built.
CBILD	There are, at the time of writing this thesis, no facilities for sorting definition tables. Sorting is at present accomplished by a combination of editor and file manager commands.

system, file manager and editor commands. Table 5.4 lists the programs from Fig. 5.3 which have NOT been implemented. The table also outlines how functions carried out by these programs are presently effected by interim measures.

5.3 Real-Time Programs in the DISCO Suite:

A 'real-time' program' is defined by Wirth [67] as one whose validity depends on its execution speed. This is a particularly good definition because it excludes from the real-time category, programs whose speed is optimised for economic or convenience reasons. This section will deal with software in the DISCO suite which operates on external stimuli and must react to those stimuli within a specific time in order to produce valid results.

5.3.1 The DISCO Executive:

The DISCO executive handles resource management of the ART, data-base integrity checks, activity header processing and initiation of real-time software modules on the basis of information stored in the ART. The latter function is carried out by the ART processor which is a part of the same program. The ART processing task may be designated a real-time computational process because it involves response to external events whose timing is not determined by the computer. Failure to respond with

sufficient speed to these events would lead to aliasing in input sampling and instability of the sampled-data control schemes. Wirth [67] gives an interesting discussion of how a real-time, computational process such as DISCO and an external real-time process, such as a chemical plant, may be treated as cooperating sequential processes for the purposes of analysis. The analysis might include graphical or pseudocode representation of the parallel processes for the identification of critical sections which should be mutually excluded.

Fig. 5.5 illustrates the top-down software structure of the ART processor. The periodic scan is initiated by the program scheduler in RTE IV. This occurs every second in the present implementation but the period itself is a DISCO variable and may be specified as required. Scanning consists of sequential processing of the contiguous activity records in the ART. Timing criteria are evaluated for activities which are marked as clock-initiated. If the criteria are satisfied, then processing of segments of the activity record is carried out.

It is an important feature of DISCO that the interface to segment processor subroutines is identical for all segment types. This means that a programmer concerned with providing a new segment or modifying one which already exists would need to know only a bare minimum about the operation of the overall software system. Since segments and their processors are simple to synthesise and edit, the

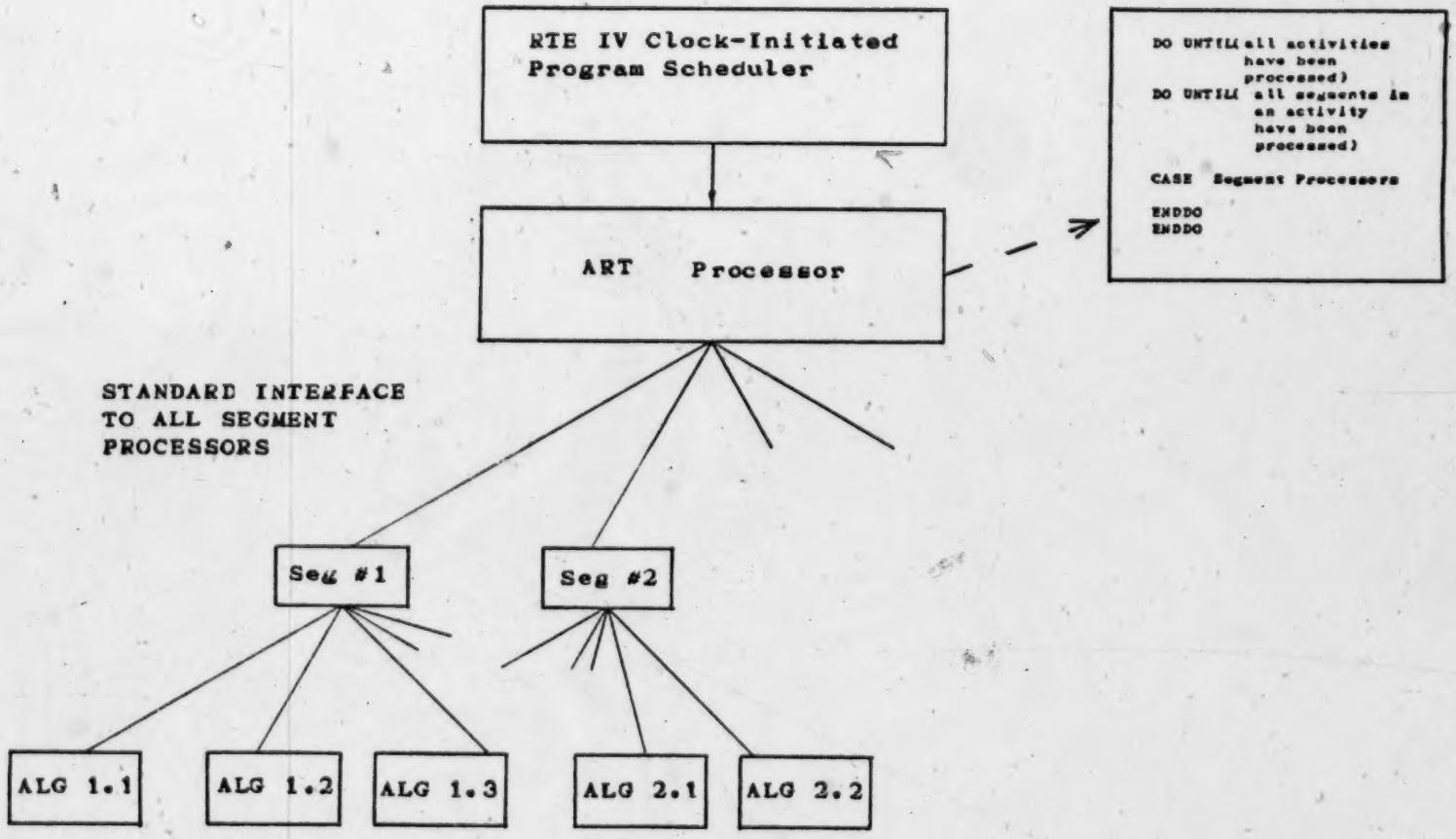


Fig. 5.5 Top-Down Processing of the ART

capability of DISCO should be constrained only by the hardware available.

5.3.2 Supervisory Programs:

Supervisory programs which may be implemented as part of the control system are informally identified as those having one or more of the following properties:

(i) programs which adjust control scheme set-points according to changing process conditions and economic objectives

(ii) programs which are initiated at greater intervals (minutes) than typical DDC periods (seconds)

(iii) programs which implement control schemes that are too complex to describe with standard ISA process and instrumentation symbols [77]

(iv) programs which are predominantly not table-driven.

Supervisory programs may be initiated as part of segment processing or they may exist entirely in higher-level software layers. Programs of the latter variety would be expected to use the data-base message processor program ACCES for fetch/edit operations on the data-base.

5.3.3 ACCES (The Data-Base Communication Processor):

Goldsack [26] describes how the use of a communication processor which is synchronised with a control program such as DISCO, ensures consistent fetch/edit operations on a real-time data-base. Synchronisation guarantees that values taken from the data-base in a single communication are obtained between consecutive scans and similarly that several parameter changes may be made within a single sampling interval.

Goldsack suggests that the implementation be effected by mutual exclusion of the control program and the communication processor. Since the processor's operation comprises little more than memory-to-memory move operations, its execution is very fast and does not hold up the control program in the case of contention. This scheme was further refined in the design of ACCES by providing for synchronisation in three modes.

(i) DISCO and ACCES are mutually excluded during any data-base operation by either program. This is the scheme originally proposed by Goldsack. (Specified by a flag in the header of the referenced activity.)

(ii) DISCO and ACCES are mutually excluded from processing the same activity record. (Specified by a flag in the header of the referenced activity.)

(iii) No mutual exclusion.

The first mode ensures that consistency is maintained when several activity records are used to implement a single control scheme. For example, in the case when a number of activities are given a group designation. The second mode is the most commonly used and assumes that the referenced activity is self-contained. This mode requires the use of two Dijkstra primitives [18], the first to govern access to the activity record and a second to govern access to a pair of pointers. The pointers specify the activities being processed by DISCO and ACCES at any given time. If either program requests an activity record which the pointers indicate is in use by the other, then the former program is suspended until the latter releases the required activity. Since activity processing takes much less time than processing of the entire ART, the risk of contention is considerably lower than in the first mode.

Since the second mode of exclusion involves overlapped allocation of two resources, namely the activity record and the pointers, consideration was given to the possibility of deadlock. Havender [31] presents several methods of resource allocation which violate the necessary conditions for a deadlock. DISCO and ACCES request and release their resources in the same order thus satisfying the terms of one of Havender's methods. It was concluded, therefore that the programs can never become deadlocked and

that the synchronisation would operate satisfactorily.

The third mode which provides unsynchronised access to the data-base is used for communication with data records which are outside the ART.

Since ACCES is only a communication processor it cannot be initiated directly by an operator. ACCES is called by other programs which need to operate on the data-base and it obtains buffers containing values and control information from such programs. Fetch and edit operations which may reference fields of arbitrary bit-length in the entire data-base are supported in this way.

5.3.4 Integrity Measures Associated with Real-Time Data-Base Manipulation:

Corruption of the data-base could lead to expensive and dangerous malfunctions of the plant connected to a DISCO control system. Considerable attention has been paid to checking of validity of real-time operations. This section will describe some integrity measures incorporated and others which may become part of the complete DISCO system.

Some features of ACCES which ensure that fetch/edit operations on the data-base are consistent have already been described in the preceding section. The use of a communication processor also enhances integrity in several other ways. ACCES checks that data-items addressed in transfer operations are within the addressed segment. It

independently also checks that the data-item is within the specified activity record. There is also scope for the use of multiple communication processors such as ACCES, each of which would be dedicated to a specific portion of the data-base. By ensuring that calling programs use the correct processor, one could also ensure that a data-area beyond that allocated to the processor would not be accidentally corrupted.

In processing the ART, the DISCO executive also makes numerous checks to ensure that it is processing genuine, uncorrupted activity records. Table 5.5 shows how the compounded effect of such checks results in a likelihood of only six in ten million that bad data will be misinterpreted as an activity record. Further integrity checks included in segment processors are expected to ensure that any accidental overwriting of the data-base can be immediately detected and remedied.

Schoeffler [48] describes a number of techniques which may be used to restart distributed processing in the event of a failure. His discussion is oriented towards the integrity of distributed data-base management schemes but also has important implications for real-time schemes such as DISCO.

Schoeffler proposes that the integrity problem should be divided into two components. The first would be concerned with integrity of operations which update data in remote processors. Within a single processor node, operations would

Table 5.5 Probability of Processing an Activity Record Corrupted by Bad Data

Probability that random data can be interpreted as an activity ID

-Activity ID must be positive.
Probability that random data will be positive is 1/2
-Activity ID has a lowest order octal digit equal to '1'.
Probability that random data will have a lowest order octal digit equal to '1' is 1/8

Probability that random data can be interpreted as an activity length

-Activity length is positive.
Probability that random data will be positive is 1/2
-Activity ends within the ART.
Assuming an ART of 1000 words, the probability that a random length will terminate the activity record within the ART is 1000/32767

Probability that random data can be interpreted as an 'activity enable' bit

-Probability that a random bit is 'on'...1/2

Probability that random data can be interpreted as consistent timing criteria

-Poll time must be greater than phase time.
Assuming a typical poll time of 10 secs the probability that random data (1 byte) will be in the range 0-9 is 0.039

Table 5.5 ...contd.

Probability that random data can be interpreted as the first segment ID

-At present valid segment ID's are in the range 1-8. Probability of random data (1 byte) being in this range is 8/255

THE PROBABILITY THAT RANDOM CORRUPT DATA IS MISINTERPRETED AS A GENUINE ACTIVITY IS THE PRODUCT OF THE ABOVE PROBABILITIES ..

..THE PRODUCT IS..0.0000006

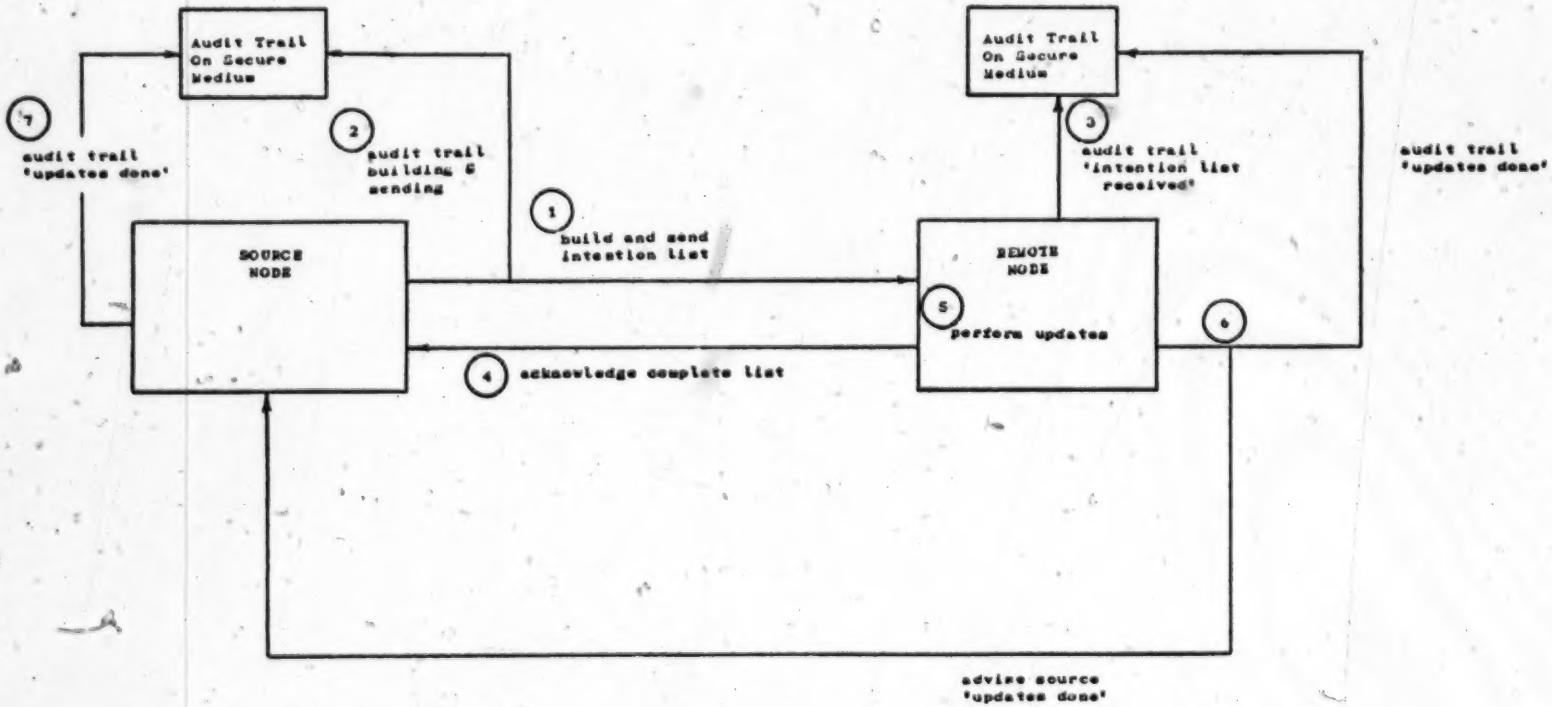
be checkpointed and critical updates of the data-base could be logged in an audit trail stored on a secure peripheral medium. A restart of the system, forced by some software or hardware failure would involve repeating all edits made since the last successfully executed checkpoint. These edits could, of course, be retrieved from the audit trail.

A scheme such as this could be the basis of a failure recovery system in DISCO. If each activity record were copied to a buffer for processing and then restored when processing of all segments was complete, the ART processor would perform just one edit operation per activity. These updates could be checkpointed thus permitting a restart in case of a failure in the manner that Schoeffler describes. Care would have to be taken to ensure that the strategy adopted achieved a suitable compromise between a heavy overhead of frequent checkpointing and slow restarts due to inadequate checkpointing. A limitation would have to be imposed that activity processing should involve only repeatable operations on systems and data outside of the DISCO data-base. The limitation is necessary because restarting an activity might involve re-executing some of the segments which were completed successfully before the malfunction occurred. For example, if a segment processor had set a positional valve output then repeating the operation would still leave the valve in the same, correct position. If, however, the valve included an incremental summation device then re-execution would double the latest

change in the valve's position.

Audit trails of changes made by the process operator would ensure that the data-base could never be in an ambiguous condition as a result of failure during processing of an update. Restart routines could ensure that all changes made since the last successfully passed checkpoint were repeated. In the case of a complete re-initiation of activity processing, all edits logged in the audit trail since the last ART backup was made could be repeated on the restored data-base.

Updating of data in remote processors may be carried out with the aid of intention lists as illustrated in Fig. 5.6. These involve a mechanism proposed by Schoeffler to guarantee the unambiguous condition of a remote data-base in the case of communication failure. The remote edit starts with transmission of the intention list which fully specifies all values and addresses required for a particular update. It is acknowledged by the recipient processor (or else it is retransmitted) which then proceeds to execute the changes. If this processing is checkpointed, the update may be restarted in the event of a failure. If the intention list itself is to be marked as an audit trail then it must be stored on a secure medium. The editing operation is carried out to completion and then acknowledged to the source processor. If this acknowledgement fails to arrive then it may be subsequently prompted by commanding a restart of the intention list in the remote module.



Actions in case of failure:

1. If audit trail indicates 'intention list not sent', rebuild send and record in audit trail.
2. If recording in audit trail fails, try again N times.
3. If recording in audit trail fails, try again N times.
4. If acknowledgement fails, re-transmit N times until received.
5. In case of failure during updates, get intention list from audit trail and restart.
6. If recording in audit trail fails, try again N times.
   If 'updates done' message to source node fails, try again N times.
7. If recording in audit trail fails, try again N times.

Fig. 5.6 Schemmatic Representation of Error Recovery Mechanisms Which May Be Used to Ensure the Integrity of a Distributed System

A computer control system differs in some respects from the general distributed network model assumed by Schoeffler. Schemes such as those described above would have to be combined with some special considerations related to restarting specific process services. An example is the restarting of control algorithms which use difference equation approximations to continuous time derivatives. These rely on the use of data points equally spaced in time and would not operate correctly if their processing was simply recommenced after a failure period. The requirement here would be for the algorithm to be returned to a start-up mode in which it would collect as many evenly spaced data-points as the order of the difference equation. Only when these had been collected could the calculation of valid outputs be resumed. It is concluded therefore, that any restart strategy would have to interact with segment processors to ensure that the special needs of the process application were met.

CHAPTER 6

EXAMPLES OF DISCO APPLICATIONS

6.1 Introduction to the Chapter:

The elements of a 'first working version' of DISCO which have been implemented as part of this thesis project have been described in Chapter 5. This chapter documents an example application used to demonstrate simple DDC and simple supervisory control using DISCO. A detailed discussion of how a more demanding, industrial control problem could be handled is also presented.

6.2 The Example Application:

A schemmatic diagram of process equipment used for instrument evaluation at the Department of Chemical Engineering is shown in Fig. 6.1. The equipment uses two pumps to circulate water in a flow cycle which includes tanks, valves and pneumatic transducers. All of the transducers and actuators can be interfaced to the HP 2240 measurement and control subsystem, thus making them accessible to DISCO.

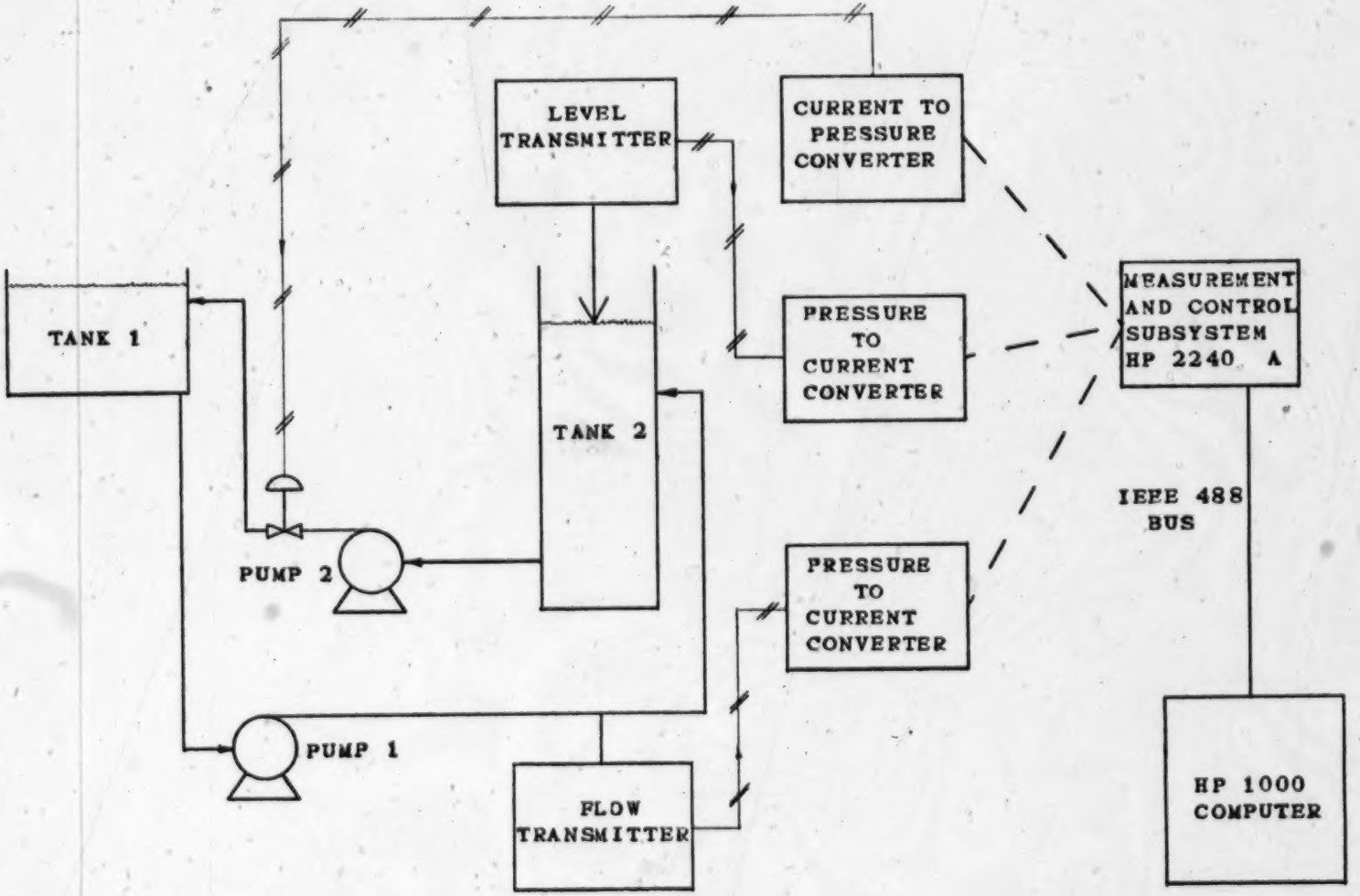


Fig. 6.1 Schemmatic Diagram of the Instrumentation Evaluation Rig Used for an Example Application.

The instrumentation shown in Fig. 6.1 implements the following two process activities:

(i) level control in tank 2 by manipulation of the outflow from tank 2

(ii) monitoring of the inflow to tank 2.

The layout and content of the activity records which define these activities are shown in Appendix A.

The first activity implements single-input, single-output feedback control of the level and facilitates the evaluation of manual, proportional and proportional plus integral control algorithms. The second activity monitors the main source of disturbance in the process, namely the inflow to tank 2. Implementation of additional algorithms would permit the latter information to be used in feedforward control schemes.

A simple supervisory program called DEMO was written to demonstrate various operational options of the two activities. DEMO also shows how process operators' console and supervisory programs may communicate with the data-base by using the communications processor ACCES. Table 6.1 shows the sequence of options exercised by DEMO and the features of the DISCO activities which are demonstrated by each.

DEMO executes the options of table 6.1 in a fixed order as shown. The program suspends itself after making the

Table 6.1 The Sequence of Control Options Exercised by the Program DEMO

Option	Description
1	-Activities enabled -Level loop on manual -Valve fully closed
2	-Display level and flowrate -Level loop on manual -Valve fully open
3	-Display level and flowrate -Level loop on auto -Prop. control Prop. Gain = 10 Setpoint = 2.875
4	-Display level and flowrate -Level loop on auto -Prop. control New setpoint = 3.868
5	-Display level and flowrate -Level loop on auto -Prop.+Integral control Prop. Gain = 10 Setpoint = 2.875 Integral Time = 20
6	-Display level and flowrate -Level loop on auto -Prop.+Integral control New Setpoint = 2.412
7	-Display level and flowrate -Exit

specified changes for a particular option and allows the operator to run the option for any length of time. The next option in sequence is initiated by issuing a command which removes DEMO from the 'suspend list'.

The data-base changes initiated by DEMO show the value of using a communications processor such as ACCES. Step 3, for example, comprises four distinct operations. Namely:

(i) set loop on automatic

(ii) set algorithm to proportional control

(iii) set proportional gain to 10

(iv) set setpoint to 2.875.

If synchronisation were not guaranteed by ACCES then operations (ii),(iii) and (iv) would have to be carried out in a preliminary edit. Only then would it be possible to set the loop on automatic and to be certain that old control algorithm specifications were not used. Since ACCES is synchronised with DISCO, all of the above operations can be carried out in one update. This may be done with complete confidence that the entire set of edits will be implemented between consecutive DISCO scans of the activity concerned.

Fig. 6.2 shows a annotated recorder plot of the level

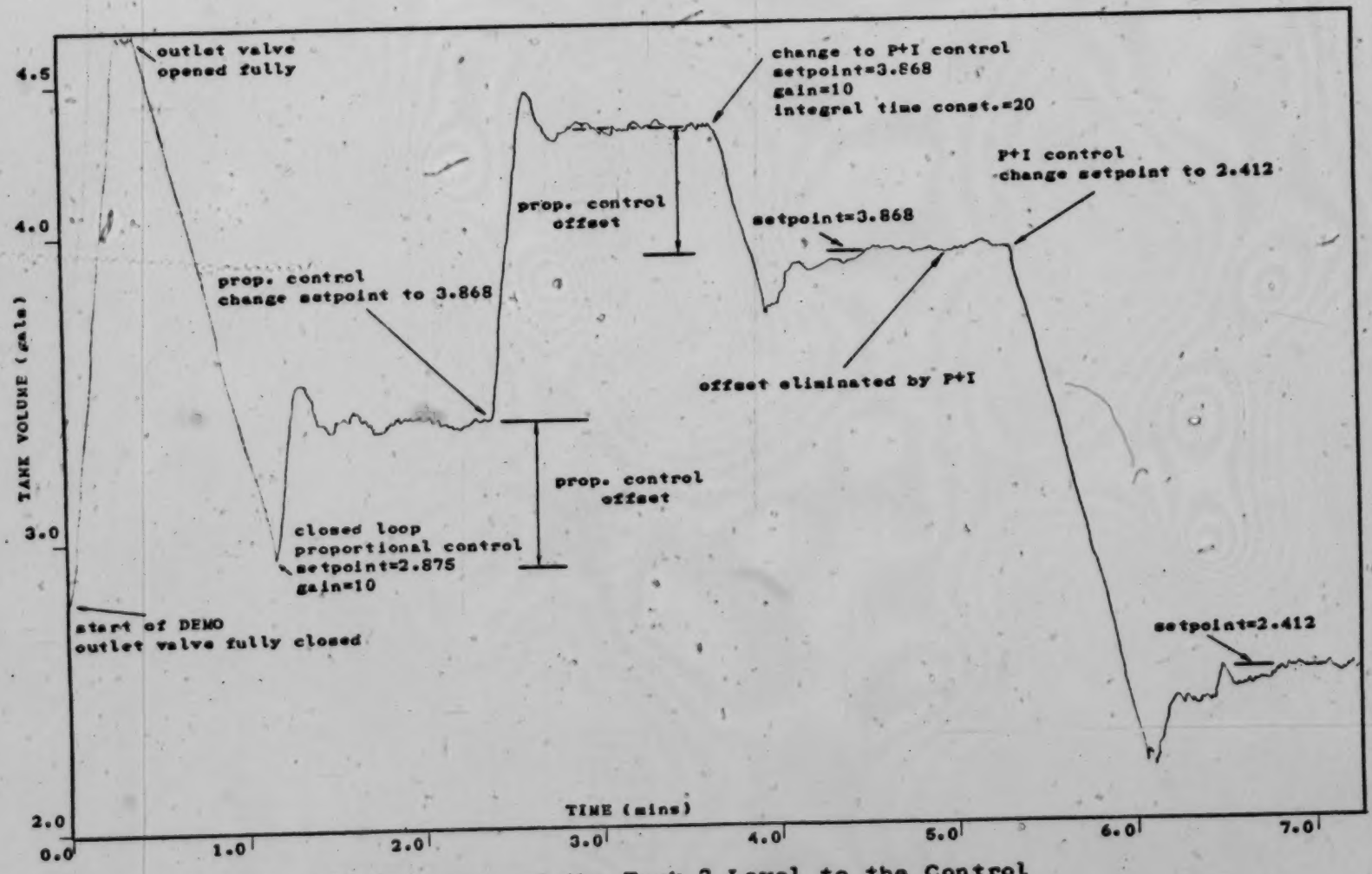


Fig. 6.2 Response of the Tank 2 Level to the Control Options Exercised by Program DEMO

in tank 2 in response to the control options of DEMO. The tank is calibrated in gallons but since it is a right, circular, vertical cylinder, there is a linear correspondence with level. The six modes exercised by DEMO are marked on the plot of Fig. 6.2. It may be seen that each mode exhibited the expected behaviour. The results may be summarised as follows.

(i) The level rose at a constant rate in response to closure of the tank outlet valve.

(ii) The level fell at a constant rate in response to opening of the tank outlet valve.

(iii) The level settled to a steady equilibrium value under proportional control. A finite offset was observed.

(iv) A change in setpoint moved the level to a new steady equilibrium value. Again there was a finite offset.

(v) Changing to proportional plus integral control eliminated the offset. DEMO showed the process variable to be within 0.16% of the setpoint.

(vi) A setpoint change under proportional plus integral

control moved the level to a new equilibrium. Once more there was no offset, the process variable was within 1.5% of the setpoint.

It was concluded that DISCO input, output and control mechanisms implemented as part of the thesis project are operating correctly. Building the data-base for the example application with the help of interactive utilities was found to be convenient and staightforward. The construction of higher-level software layers for supervisory and console services also posed no difficulties. Additionally it was concluded that the communications processor ACCES provided better efficiency in consistent fetch/change operations than schemes permitting free data-base access for all programs. ACCES also constituted a safe and clean interfacing mechanism which significantly improved the integrity of data-base update operations.

6.3 Design and Implementation of DISCO Segments:

Three segment types have been designed and implemented for use with DISCO as part of the thesis project. These provided the means for implementation of the example application of section 6.2. The segments define input, control and output operations in a variety of modes and configurations. The segment formats are shown in tables 6.2, 6.3 and 6.4.

Table 6.2 Input Segment Format Designed and Implemented for Operation in the DISCO Data-Base

Word	Description	Field Length(bits)
1	Segment ID	8
	Segment length	8
2	Segment enable/disable	1
	Input operational/non-op.	1
	Input from process yes/no	
	Input from buffer yes/no	1
3	Zero	32
4		
5	Span	32
6		
7	Label	32
8		
9	Input raw value (1)	16
	.	
	.	
	.	
N+8	Input raw value (N)	16
N+8+1	Input address (1)	16
	.	
	.	
	.	
2N+8+1	Input address (N)	16

Table 6.3 Control Segment Format Designed and Implemented for Operation in the DISCO Data-Base

Word	Description	Field Length(bits)
1	Segment ID	8
	Segment length	8
2	Control enable/disable	1
	Loop startup yes/no	1
	Reset windup flag	1
	Algorithm is subroutine /program	1
	Setpoint change permit yes/no	1
	-- spare 3 status bits -	
	Algorithm ID	8
3	Setpoint	16
4	Proportional constant	16
5	Integral time	16
6	Derivative time	16
7	Integral continued sum	16
8	Input pointer (1)	16
	.	
	.	
	.	
N+7	Input pointer (N)	
N+7+1	Output pointer (1)	16
	.	
	.	
	.	
M+N+7+1	Output pointer (M)	16

NOTE : The input and output pointers are not needed when operating with the data stack. Since this is generally the case, the segment is normally just 7 words long.

Table 6.4 Output Segment Format Designed and Implemented for Operation in the DISCO Data-Base

Word	Description	Field Length
1	Segment ID	8
	Segment length	8
2	Segment enabled/disabled	1
	Output operational /non-op	1
	Output positional /incremental	1
	Output to process interface yes/no	1
	Output to buffer yes/no	1
	Output on manual yes/no	1
3	Output top limit	16
4	Output bottom limit	16
5	Output rate limit	16
6	Zero	32
7		
8	Span	32
9		
10	Label	32
11		
12	Output value (1)	16
	.	
	.	
	.	
N+11	Output value (N)	16
N+11+1	Output address (1)	16
	.	
	.	
	.	
2N+11+1	Output address (N)	16
2N+11+1 +1	Output origin (1)	16
	.	
	.	
	.	

Table 6.4 ...contd.

3N+11+1 +1	Output origin (N)	16

NOTE : The output origin address is not needed when operating with the data stack. Since this is generally the case, the segment is normally just 2N+11+1 words long. (Where N is the number of outputs)

While the segments have been structured to define a wide variety of DISCO operations it is by no means expected that they will be adequate for all future applications. It is very strongly emphasised that the formats shown in tables 6.2, 6.3 and 6.4 do NOT represent the inherent characteristics of DISCO with respect to input, control and output functions. The standard interface between DISCO and segment processor routines and the structure of the data-base builder allow any of these segment formats to be quite simply altered or replaced. New segment types may be added just as simply and thus any change in hardware capability or any change in the nature of the application may be handled.

The input segment of table 6.2 has been designed to handle any number of inputs using any number of retrieval algorithms with random addressing of the source device. The data-acquisition routine for a particular input is specified as part of the address word. While the segment is sufficiently general purpose to be capable of handling almost all input services, there will be a number of functions which will be handled more efficiently by a modified format. The following are some examples.

(i) Input of data-items which do not fit in a single word. Such data might be binary-coded-decimal, 2-word floating point or multiple precision. In these cases an address for each input word is not necessary and the operation would be handled more

efficiently by a segment format which used just one address for each multiple-word entry.

(ii) Input of a number of data-items from sequential addresses in a single device. In this case only the first address and the number of transfers need to be specified. The format of table 6.2 would therefore be inefficient in providing addresses for all the inputs when most could be assumed by convention.

The above examples show limitations which may be overcome by very minor modifications and additions. If it ever becomes necessary to acquire a large number of inputs in an unusual mode, DISCO will still be capable of handling the problem because it will be possible to generate a special purpose segment to perform the service.

The control segment of table 6.3 is clearly oriented towards single-input, single-output feedback control. It is important to include a segment which defines this operation efficiently since even when advanced control schemes are used there will still be many loops of this nature. A very general segment could simply specify the control algorithm and a parameter vector. The format and length of the parameter vector would be dependent on the choice of algorithm. This type of segment design would clearly be capable of defining all conceivable control actions. The

scheme does, however, have significant disadvantages for the average single-variable application since the builder would no longer prompt the user for algorithm parameters by name.

The output segment of table 6.4 has a format similar to that of the input segment. Since the segment organisation supports random access to process outputs, similar limitations and advantages to those discussed for the input segment apply. Summarising, the segment is capable of handling almost any output service but some types of output, particularly those where many transfers are made as part of a single operation, would be handled more efficiently by a customised format.

6.4 An Industrial Advanced Control Application:

It has been stated that DISCO has been designed to handle advanced control algorithms such as those used in university research and sophisticated industrial applications. This section will describe how a difficult industrial control problem can be simply and conveniently handled. While the problem is hypothetical, it is hoped that the description of DISCO in the preceding chapters and operation of the example application will persuade the reader that the solution presented would be a valid one.

6.4.1 Description of the Example:

Decoupling control of distillation columns is one of the most widely accepted advanced control techniques to be used in chemical process plants (Wade [80]). The example which will be presented here is taken from Shinskey [54], where it is introduced as a genuine industrial case study. The decoupling control is required to eliminate interactions caused by poor process design rather than to improve regulation or response. The example is nevertheless a valid one for the purposes of this thesis.

Fig. 6.3 shows the interacting system of two parallel reboilers on a distillation column. The control objectives are:

(i) regulate the bottoms composition as indicated by tray temperature T

(ii) regulate the reboiler level L

(iii) regulate the reboiler level L'.

The available control variables are:

(i) reboiler heat input Q

(ii) reboiler heat input Q'

(iii) bottoms flowrate B.

SYMBOLS :

T	Tray temperature from which composition is inferred.
L, L'	Reboiler levels.
Q, Q'	Reboiler heat inputs.
B	Bottoms product flowrate.

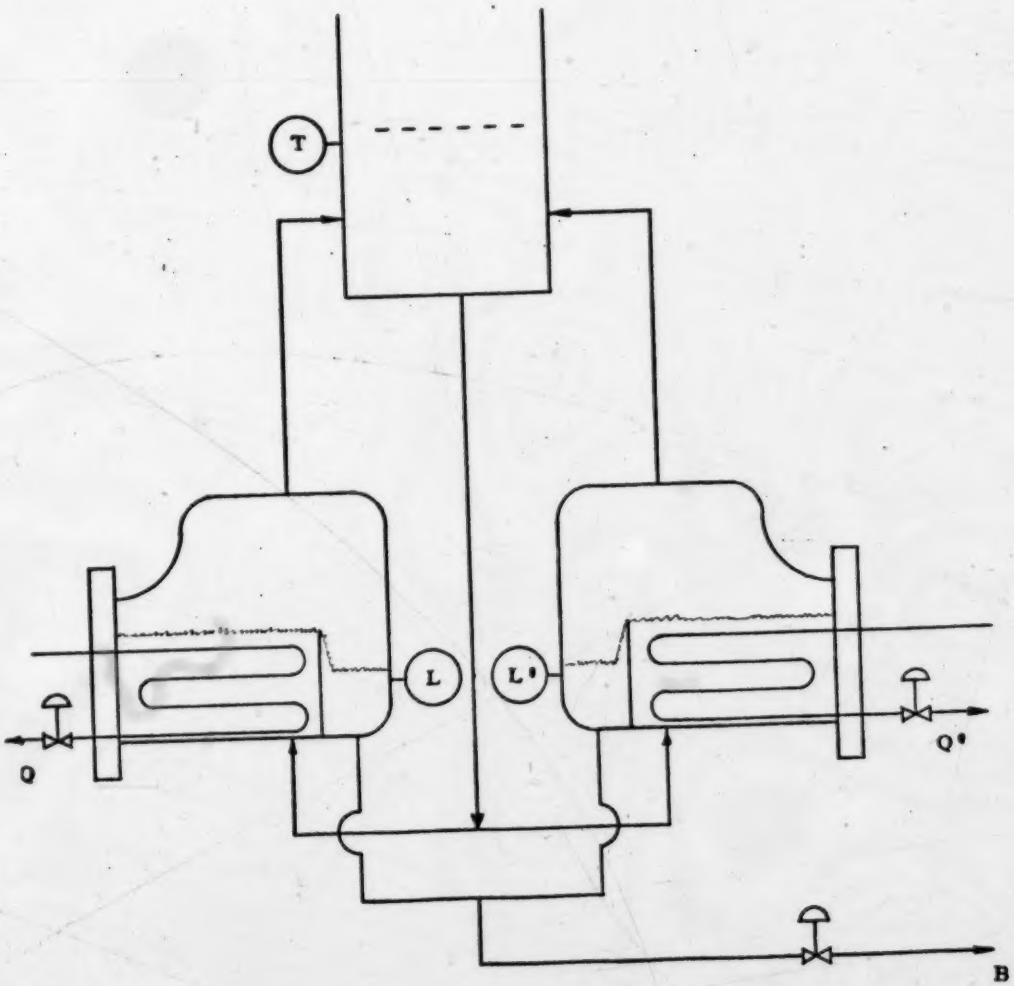


Fig. 6.3 Example of Interacting Reboilers Taken from Shinskey [54].

Shinskey analyses in some detail, the nature of the interaction. Since the discussion here is not concerned with control scheme design, the analysis is not presented. It is however, of interest to briefly consider why interactions exist. It is clear from inspection of Fig. 6.3 that no satisfactory pairing of inputs and outputs is possible. The product flowrate B, equally affects both levels and hence cannot be used to control either of them independently. Both heat inputs affect all three of the output variables. As a result, they cannot be used for level control because the composition would then become a function of level setpoint.

Shinskey presents decoupling control as a simple cure for the poor process design which caused such catastrophic interactions. It is obviously possible to remedy the situation by restructuring the piping configuration of the reboilers but this is much less convenient than a simple modification of the control algorithm. The decoupling control is based on synthesis of new input and output variables from linear combinations of the real process inputs and outputs.

Shinskey describes the proposed scheme as follows:

> 'The level signals were averaged and used to control the draw-off valve. The temperature controller manipulated both heat-input valves, biased by the differential level controller'.

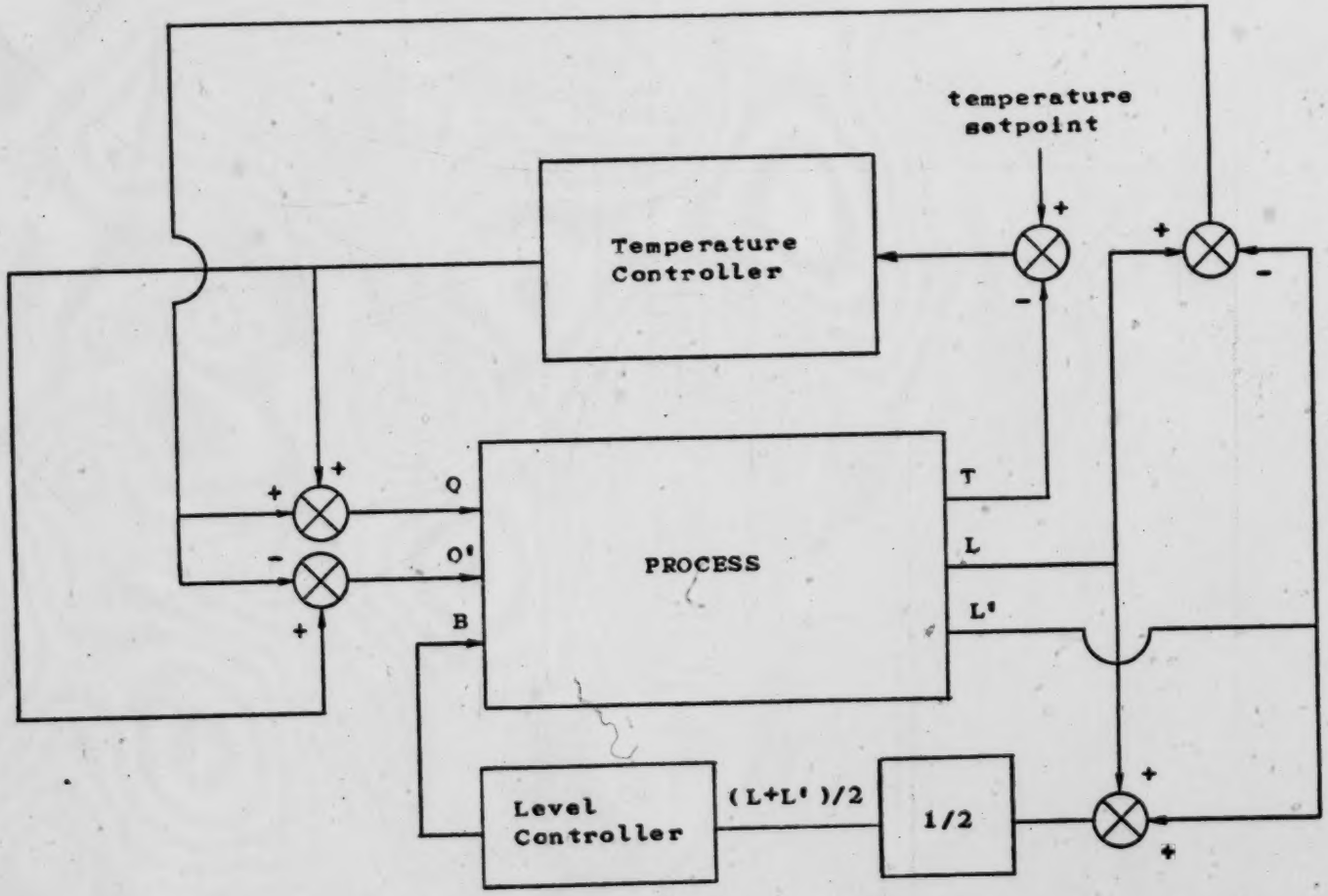


Fig. 6.4 The Decoupling Control Scheme
Proposed by Shinskey [*].

Fig. 6.4 shows the control scheme in block-diagram form. Shinskey presents an interaction analysis for the system under control of the decoupling scheme. As previously stated, the control scheme design is not relevant to this thesis and hence the analysis will not be discussed. It is however, possible to see by inspection that the system is likely to behave in a more satisfactory manner. The bottoms flowrate is now used to control the combined holdup of the two reboilers. This should be no more difficult than level control with a single tank unit. Actual level regulation is effected by eliminating the level difference between the two tanks. Although this is done by manipulating heat inputs, the total heat supplied during any level control action is constant and hence the bottoms composition should remain unaffected. Composition regulation is carried out by equal, symmetrical manipulation of the heat inputs. There is an interaction with the reboiler holdup control but this appears as a load to the level control system and hence the system is not destabilised.

6.4.2 A Proposed DISCO Activity Configuration:

A DISCO implementation of the control scheme of Fig. 6.4 would require only an arithmetic segment in addition to features which already exist. Such a segment would be useful for many operations and will probably be made a part of DISCO during continuing work in the near future. A possible

format for such a segment is shown in Appendix B.

It is important to note that the discussion here relates to true DDC, table-driven operation of the scheme. Clearly it is possible with almost any computer control system to write a supervisory program to perform the desired functions. The advantages of a DDC implementation are presented below.

(i) The control scheme is implemented with existing constructions and no additional programming is necessary.

(ii) The scheme is defined in a standard manner in the data-base and all the advantages of table-driven processing are available. (These advantages were described in Chapter 4).

(iii) Standard console support designed to work with the data-base may be used.

(iv) Standard data-base utilities may be used for specification of the control scheme parameters.

The linear equations which must be specified in the arithmetic segment and which need to be evaluated as part of the control scheme are:

$L'' = L - L'$ ...6.1

$Q = x + L''$ ...6.2

$Q' = x - L''$ ...6.3

$y = (L + L')/2$ ...6.4

(where x is the output of the temperature controller and y is the input to the level controller).

The specification of these equations in the arithmetic segment is presented in Appendix B.

Fig. 6.5 shows a possible activity configuration which would handle the scheme. The method of directed labels as described in section 4.3, would be used for inter-segment communication. The use of pointers specified by the directed labels would allow the following inter-segment accesses to be made.

(i) The control segment processor would fetch the tray temperature value from the preceding input segment.

(ii) The control segment processor would store the temperature controller output (x) in the data area of the arithmetic segment.

(iii) The arithmetic segment processor would fetch the levels L and L' from their respective input

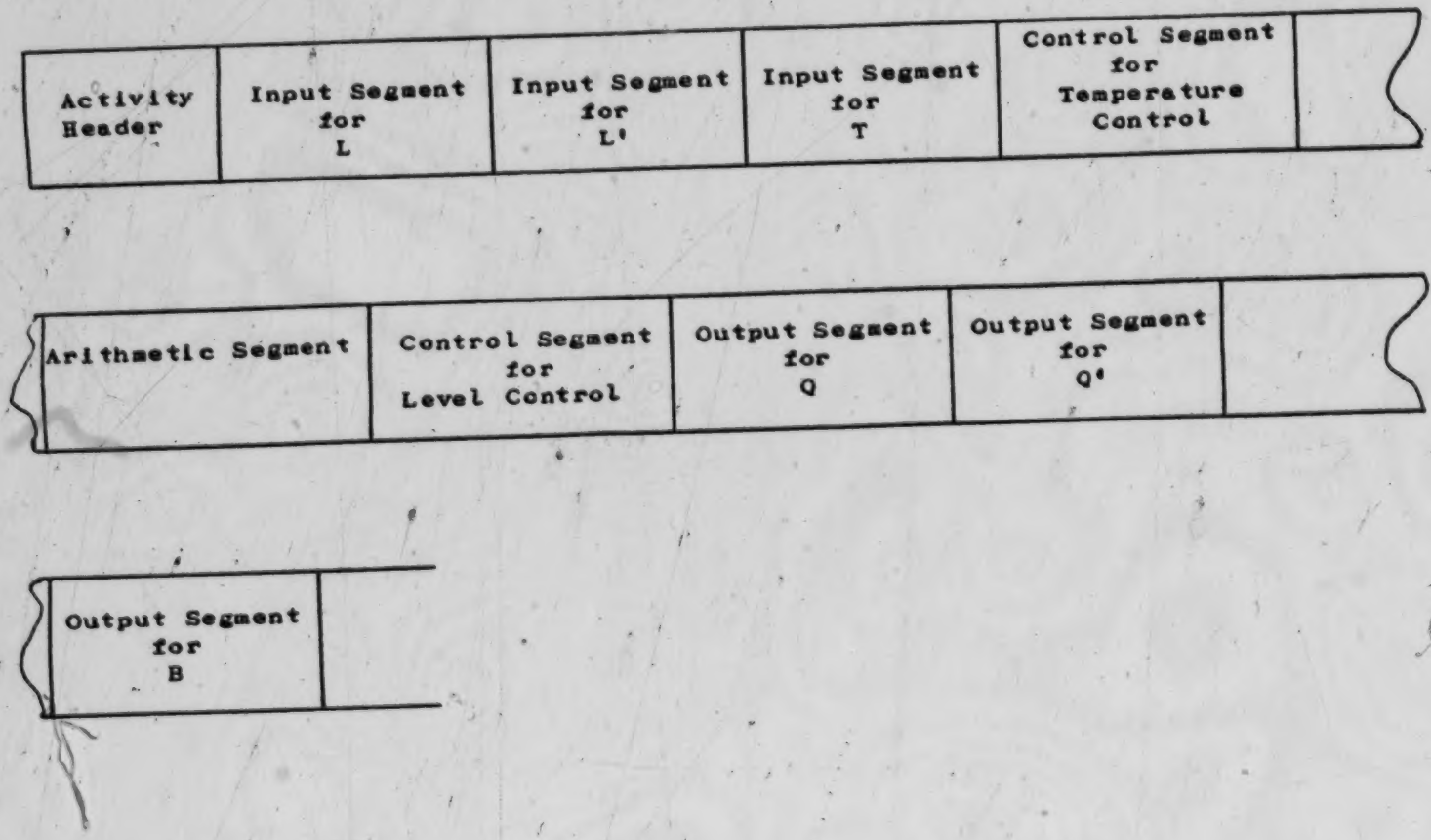


Fig. 6.5 A Proposed Activity Configuration for the Decoupling Control Scheme.

segment storage locations.

(iv) The control segment processor would fetch the level controller input (y) from the arithmetic segment data area. It would then store the computed output in the output segment for B.

(v) The output segment processor would fetch the values of Q and Q' from the data area of the arithmetic segment.

The preceding discussion has presented a simplified approach to a real DISCO implementation of Shinskey's control scheme. The purpose has been to persuade the reader that DISCO could be used to implement the scheme as part of standard table-driven, DDC operation. 'Real plant' aspects which would also have to be considered in a full design study are as follows:

(i) incorporation of filter segments to smooth noise-corrupted inputs

(ii) incorporation of alarm segments to define action in case of out-of-limits conditions

(iii) addition of gain and offset terms in equations 6.1, 6.2, 6.3 and 6.4 to facilitate tuning of the

algorithm.

CHAPTER 7

CONCLUSIONS AND CONTINUATION OF THE PROJECT

7.1 Conclusions of the Thesis Project:

A design for a distributed digital computer control system has been completed as part of the thesis project. This work has included a study of the general principles of digital computer control of engineering processes, specification of design objectives for the DISCO scheme and a partial implementation of the scheme on a system of HP 1000 computers. The expansion and development of the implementation is a continuing project in the Department and good progress is being made towards a full realisation of the DISCO design.

As a result of the work carried out in the thesis project, a number of innovative features have been incorporated in a very general computer control system. The development of these properties in the design study and their justification in the partial implementation has been the main contribution of this project. This section will attempt to summarise the new and distinguishing features which are intended to make DISCO more powerful than previous

schemes.

The flexibility of the DISCO scheme is attributable, to a large extent, to the data-base design. The activity record which is the basis of table-driven processing may contain data which define any operation of which the computer hardware is capable. These operations, defined by segments, may be carried out in any order and with any number of repetitions as specified by the arrangement of the segments themselves. The use of non-ART tables facilitates the logical referencing of instrumentation and buffers. As a result DISCO is independent of instrumentation protocols and network software. Table 7.1 summarises distinguishing and innovative features of the DISCO data-base design.

The operation of the DISCO software has been successfully demonstrated by the part-implementation. The interactive data-base building utilities were exercised comprehensively during the development of segment formats for the 'first working version'. The specification of data-base table structure and format in disk-based working tables was a particularly convenient feature of the builder. This feature allowed numerous modifications of the data-base tables to be made without any changes to the building utilities. Table 7.2 summarises distinguishing and innovative features of the DISCO data-base building utilities.

The operation of the real-time DISCO control software has been demonstrated by the example described in Chapter 6.

Table 7.1 Distinguishing Features of the DISCO Data-Base Design

(i) The data-base records which govern table-driven processing in DISCO are based on the process activity. This contrasts with most commercial DDC systems which use conventional 'feedback control loops' as a record basis.

(ii) A variety of operations defined by segments of the activity record may be incorporated in a process activity.

(iii) Segments may be variable-length and may also have a variable-format.

(iv) Non-ART tables contain information which may be accessed during activity processing. One of the applications for this type of table is to allow logical referencing of buffers and instrumentation.

(v) A memory stack is used as one method of passing data between processing operations defined by successive segments. It has been found that a stack is particularly convenient when handling 'input-calculation-output' mechanisms such as those common in process control structures.

Table 7.2 Distinguishing Features of the DISCO Data-Base Building Utilities

(i) Addition of new types of DISCO data-base tables or modification of existing types does not require the alteration of any program code. This is because the structure and format of all the data-base tables is stored in disk-files which themselves may be updated by the building utilities.

(ii) Building of DISCO data-base tables is fully interactive. Specification of data-item values is carried out in a 'fill-in' the-blanks' mode.

(iii) The conversational table-specification results in a user-oriented source version which may be edited by the building utilities. Modifications to the source version do not, therefore, require rebuilding of the table.

(iv) Provision for handling of mnemonic process variable names and symbolic labels will simplify construction of links between data-base records. When the symbol utilities are complete, such links should be as easy to handle as labels and variable names in a FORTRAN program.

Although the part-implementation was restricted to an in-memory data-base and single processor operation, the main features of DISCO real-time software could still be rigorously tested. The synchronisation between the DISCO executive and the communications processor ACCES was observed to be working well. The example application demonstrated how the use of a communications processor facilitated powerful and concise process operator commands which were executed in a controlled, predictable manner. Table 7.3 summarises distinguishing and innovative features of the real-time DISCO software.

7.2 The Continuing Project:

As stated in the previous section and in preceding chapters the implementation of a complete DISCO system is a continuing project in the Department of Chemical Engineering. This section discusses the major areas of outstanding work and describes how the development of each area will contribute to the full realisation of the DISCO design.

7.2.1 Future Work in Expanding the Data-base Building Utilities:

The most important task in the expansion of the data-base utilities is the development of the DOPE program

Table 7.3 Distinguishing Features of the DISCO Real-Time Software

(i) Synchronised access to the real-time data-base for fetch/update operations is guaranteed by the communication processor ACCES.

(ii) A standard interface between the DISCO executive and segment processor subroutines has been provided. This feature facilitates simple addition of new segment types and their processors. The author of a processor subroutine is able to deal with a single documented interface and would never have to modify any existing code to incorporate his routine.

(iii) The basic period of cyclic ART processing may be set arbitrarily when scanning is initiated. This is one of many features which allow DISCO to be adapted to the capabilities of specific computer hardware in a particular working environment.

described in section 5.2.4. This program will be an executive which will guide the user in a conversational mode, through the process of building, editing and loading a data-base. The executive will also manage all disk files associated with building and maintain a load map to ensure that new tables do not overwrite existing records.

The completion of DOPE has been given high priority in the continuing project because of the organisational improvements it will effect. These improvements will permit new users to safely experiment with the data-base building utilities without fear of overwriting disk-files or the real-time data-base. In some cases DOPE will make the use of disk-files transparent to the user. The automatic allocation of activity ID's and the automatic calculation of activity record lengths will make data-base building and real-time operations more reliable.

When the Data-Base Operations Executive is implemented, attention will be given to completing support for symbolic references during conversational building. The builder and editor program SBILD already accepts users' variable names for data-item values and directed labels. To make this feature fully operational, DOPE will have to create files for storage of symbol tables and a directory of these files will have to be maintained. An additional utility will be required in the suite to build the tables and to resolve symbols before loading.

7.2.2 Future Work in Development of Process Operator Console Support:

It has been stated in section 5.1 that support for industrial standard operators' consoles will not be designed or implemented in the DISCO project. Support of this type would constitute a higher-level software layer which, if required, could always be simply interfaced by means of the communications processor ACCES.

There is however, a requirement for simple process operator utilities which are based on conventional ASCII computer terminals. Development of these utilities is currently in progress in the Department. When complete, the process operator support software will facilitate specification of fetch and change operations on the data-base in terms of mnemonics. The utilities will also allow user-oriented data-formats such as decimal integer, decimal floating point or ASCII characters to be used. The console commands will, where possible, make hardware-oriented data-item addressing transparent to the user. As a result, word address, bit position and field-length will have to be specified in only very exceptional circumstances.

It is intended that despite restriction of console support to computer terminals the man-machine interface will be flexible and human-oriented. Communication with the operational data-base should be as simple as manipulation of the disk-based source versions by building utilities. The process operator console software will be adequate for

operation in the university environment and will demonstrate how a special purpose industrial standard console could be interfaced to DISCO.

7.2.3 Future Work in Development of New Segment Types:

The input, output and control segment designs implemented as part of the thesis project have been described in Chapter 6. Extensive interfacing of pilot-plant applications will require at least alarm and filter segments to be implemented. Other segment types such as the arithmetic segment described in Appendix B will be added as DISCO is expanded to meet a variety of operational needs.

In some instances compromises will be made between the use of segments and non-ART tables. This situation will arise where the segment contains data which is associated with an activity but is not processed each time the activity is polled. A typical example of such usage would be a process display segment which stored addresses of key variables in the activity for display purposes. Addresses such as these could alternatively all be stored in a single non-ART table. The organisation of the data-base will be influenced in this respect by:

(i) the circumstances of normal access to the data concerned

(ii) memory and processing overheads

(iii) clarity of the organisation from the point of view of the user

(iv) compatibility with use of peripheral storage media for less frequently accessed data.

7.2.4 Future Work in the Development of the DISCO Real-Time Software:

The DISCO control software is at present limited to a single-processor, in-memory scheme using the HP 2240 subsystem as a process interface. As a part of future work DISCO will be extended to handle overlay ART's from local and network accessible storage media. A priority in this respect will be the implementation of overlay swapping to and from the local disk. This will allow activity records for slower activities to be stored on disk and thus extend the capacity of the control system.

The I/O facilities proposed for the DISCO scheme were presented in section 4.4.3. Future work in this area will involve the implementation of buffer transfer programs for handling local and network I/O. Both synchronous and asynchronous buffer operations will be required and there may also be a need for special scans of the ART to be triggered by arriving data. This work will expand the

capabilities of DISCO to permit the use of 'front-end', process interface microcomputers, multiple instrumentation buses and network process I/O.

7.2.5 Future Work in Expansion of DISCO to Multiprocessor Operation:

The implementation of DISCO on a distributed network of digital computers will enhance the flexibility, reliability and capacity of the control scheme. This work will involve:

(i) extension of the I/O facilities for network operation as discussed in the previous section

(ii) design and implementation of software to perform integrity checks and error recovery during communication with remote data-bases

(iii) extension of data-base building and process operators' console utilities for remote operations.

Modification of the data-base building and process operator console utilities for distributed operations should be mainly concerned with design of a user interface. The communications processor ACCES may, in its present form, be

used for fetch/edit operations from a remote node. It will therefore, only be necessary to implement a higher-level program capable of issuing the remote scheduling call.

A similar approach could be taken to implement remote data-base loading. Activity records and non-ART tables could be loaded by a program scheduled by local or remote data-base building utilities in much the same way as ACCES is scheduled by the console utilities. The ACCES program itself could, in fact, be used but since it is designed to edit a small number of data-fields in a flexible manner its operation as a loader would not be very efficient.

It is expected that use of DS/1000 software and the associated high-level language network interface will greatly simplify synthesis of reliable information transfer mechanisms. Integrity and error recovery at the message level will be handled by the network software. This means that attention need only be given to ensuring the integrity of complete updates consisting of several messages. Restarting of operations following a failure in a particular node will have to allow for the special requirements of restarting control algorithms as discussed in section 5.3.4.

It is likely that edit operations carried out by the ACCES program and loader will be recorded in an audit trail. The data-base can then be re-synthesised from a back-up copy if any loss of data occurs as a result of computer hardware failures.

REFERENCES

(1) Alden, R. M., 'Microcomputers for Remote Intelligent Monitoring & Control', ISA Annual Conference, Houston, 1976

(2) Bairstow, R., Barlow, J., Waters, M. & Watson, J., 'A Distributed Micro-Computer System for Digitising Machines', Procs. of IEE Int. Conf. on Distributed Computer Control Systems, Birmingham, England, Sept. 1977

(3) Bates, D.G., 'PROSPRO / 1800', IEEE Trans. IECI, Vol. IECI-15, No. 2, Dec. 1968

(4) Berka, V. F., 'Documentation for Program PRDOC', DACS Centre, Dept. of Chem. Eng., University of Alberta, 1978

(5) Bristol, E. H., 'Designing and Programming Control Algorithms for DDC Systems', Control Engineering, Jan. 1977

(6) Bristol, E.H., 'The organizational Requirements of the Process Control Distributed System', Proceedings of the Joint Automatic Control Conference, Philadelphia, October 1978

(7) Brown, P. J., 'Programming & Documenting Software Projects', Comp. Surveys, Vol. 6, No. 4, Dec. 1974

(8) Brukbaker, R. H., 'An Intelligent Peripheral for Measurement and Control', HP Journal, July 1978

(9) Chandra, S., 'A Communication Network for Distributed Data Acquisition and Control in Industrial Plants', IEEE Trans. Industrial Electronics and Control Instrumentation, Vol IECI-25, No. 3, Aug 1978

(10) Cromwell, N. O., Griem, P. D., 'Software for the FOX 1 Interactive Process Operator Console', Preprints of IFAC/IFIP 1st Symposium on Software for Computer Control, Tallinn, USSR, May 1976

(11) Cronhjort, B. T., 'APG/7 - Application Program Generator for the IBM System/7', Preprints of IFAC/IFIP 1st Symposium on Software for Computer Control, Tallinn, USSR, May 1976

(12) 'DACS Centre', Department of Chemical Engineering, University of Alberta, January 1974 (to be revised)

(13) Dallimonti, R., 'Distributed Control Puts the "Smarts" Where the Action Is', Instruments & Control Systems, Vol. 50, No. 6, June 1977

(14) Daneels, A., Mead, P., 'Implementation of a Data Base in a Distributed Computer System for Real-Time Accelerator Control', Procs. IEE Int. Conference on Distributed Computer Control Systems, Birmingham, England, Sept. 1977

(15) Davies, D.W., Barber, D.L.A., ' Communication Networks for Computers ', Wiley, 1973

(16) DDC Manual, DACS Centre, Dept. Chem. Eng., University of Alberta, April 1977

(17) Diehl, W., 'Software for Industrial Computer Control - A Review', Preprints of IFAC/IFIP 1st Symposium on Software for Computer Control, Tallinn, USSR, May 1976

(18) Dijkstra, E.W., ' Cooperating Sequential Processes in Programming Languages ', (F.Genuys, Ed.), Academic

Press, New York, 1968

(19) Dijkstra, E.W., 'Notes on Structured Programming', Technical University, Eindhoven, The Netherlands, 1970

(20) Dodd, G.C., 'Elements of Data Management Systems', Comp. Surv., Vol. 1, No.2, 1969

(21) Engleberg, G.P., Howard, J.A., Mellichamp, D.A., 'A Real-Time Simulation of a Hierarchial Computer Network for Process Control', Simulation, Vol. 30, No. 2, Feb 1978

(22) Entwistle, A. G., Oldfield, N. J, 'A High Level Interactive Software System for Real-Time Multiprocessor Applications on Modern Power Plant', Procs. IEE International Conference on Distributed Computer Control Systems, Birmingham, England, 1977

(23) Fisher, D. G., Seborg, D. E., ' Multivariable Computer Control - A Case Study', American Elsevier, New York, 1976

(24) Fisher, D.G., 'Experience with Computer Networks in a University Environment', Preprints of 71st Annual

AIChE Conference, Miami, Florida, Nov 1978

(25) Freeman, P., ' Software Systems Principles', SRA 1975

(26) Goldsack, S. J., 'The Network Microprocessor as a Cooperating Sequential Process', Procs. IEE International Conference on Distributed Computer Control Systems, Birmingham, England, 1977

(27) Gordon, D., Spencer, R. D., 'Economic Aspects of Distributed Systems', Procs. IEE International Conference on Distributed Computer Control Systems, Birmingham, England, 1977

(28) Haberman, A. N., ' Introduction to Operating System Design', Science Research Associates 1976

(29) Hanson, V., Jaaksoo, U., 'Software for an Advanced DDC System', Preprints of IFAC/IFIP 1st Symposium on Software for Computer Control, Tallinn, USSR, May 1976

(30) Hanna, F. K., Hinton, O. R., Dimond, K. R., Alianak, T., Myers, P. J., Dawson, C., 'A Microprocessor Network for Distributed Industrial Control', Procs. of IEE Int. Conference on Distributed Computer Control Systems, Birmingham, England,

1977

(31) Havender, J. W., 'Avoiding Deadlock in Multitasking Systems', IBM Sys. J., Vol. 7, No. 2, 1968

(32) Howard, D.W., Insley, M.G., 'The Medway Water Board Telemetry and Telecontrol System', Water Services, March 1974, April 1974

(33) 'Installation Management - An Introduction to Structured Programming in FORTRAN', IBM Technical Publication GC20-1780-0, July 1977

(34) Jenkins, D. G., 'Microprocessors and Process Control: More Than a Marriage of Convenience?', Procs. of IEE Int . Conference on Distributed Computer Control Systems, Birmingham, England, 1977

(35) Jervis, P., 'Communication and Software Elements for Distributed Control Systems', Procs. IEE Int. Conference on Distributed Computer Control Systems, Birmingham, England, 1977

(36) Johnstone, R.M., 'Hyperstable, Model Reference Adaptive Control Systems', M.Sc. Thesis, University of Alberta, 1979

(37) Klaiss, D. E., 'Firmware Intelligence for Measurement and Control Processing', HP Journal, July 1978

(38) Lalive d'Epinay, T., 'First Steps in Standardization of Real-Time Operating Systems', Preprints of IFAC/IFIP 1st Symposium on Software for Conputer Control, Tallinn, USSR, May 1976

(39) Lewis, J. W., Davis, J. E., 'Development of a Microprocessor - Based Clinical Analyzer Controller', ISA Annual Conference, Houston, 1976

(40) Marsh, B. E., 'A Distributed Processing System for the Electrical Industry', Procs. of the IEE Int. Conference on Distributed Computer Control Systems, Birmingham, England, Sept. 1977

(41) Martin, J., ' Telecommunications and the Computer', Prentice Hall 1969

(42) Metcalfe, R.M., Boggs, D.R., 'Ethernet: Distributed Packet Switching for Local Computer Networks', Comm. ACM, Vol. 19, No. 7

(43) Misurewicz, P., 'Simple Real-Time Executive for Small Minicomputer', Preprints of IFAC/IFIP 1st Symposium on Software for Computer Control,

Tallinn, USSR, May 1976

(44) Pearkins, J., 'FMT Reference Manual', Computing Services, The University of Alberta, Edmonton, 1976

(45) Pracht, C. P., 'A distributed Microprocessor-Based Control System Programmable by the Process Engineer', ISA Annual Conference, Houston, 1976

(46) Raphael, H. A., 'Single-Chip Computer Offers Maximum Flexibility', Canadian Electronics Engineering, Feb. 1977

(47) Schoeffler, J. D., 'Performance of Distributed System Architectures', IFAC/IFIP Workshop on Real-Time Programming, Marlehamn/Aland, Finland, 1978

(48) Schoeffler, J. D., 'Software Architecture for Distributed Data Acquisition and Control Systems', Preprints of Seventh Triennial World Congress of IFAC, Vol. 1, P. 641, Helsinki, Finland, 1978

(49) Schoeffler, J. D., Keyes, M. A., 'Hierarchial Language Processing', Preprints of IFAC/IFIP 1st Symposium on Software for Computer Control, Tallinn, USSR, May 1976

(50) Seraji, H., Tarokh, M., 'Design of PID Controllers for Multivariable Systems', Int. J. Control, Vol. 26, No. 1, July 1977

(51) Shah, S.L., 'The Role of Eigenvectors and Eigenvalues in Multivariable Control System Design', Ph.D. Thesis, University of Alberta, 1977

(52) Shaw, A. C., ' The Logical Design of Operating Systems', Prentice Hall 1974

(53) Shepherd, A. J., 'A British Example of Distributed Computing', Datamation, March 1978

(54) Shinskey, F.G., ' Distillation Control ', McGraw Hill 1977

(55) Sloman, M. S., Penney, B. K., Marsland, T. A., 'A Communication Protocol for Distributed Microprocessors', Procs. IEE Int. Conference on Distributed Computer Control Systems, Birmingham, England, 1977

(56) Smart, J. D., Wood, B. M., 'The CORAL Language and Software Development for Computer Control Systems in the Future', Procs. of IEE Int. Conference on Distributed Computer Control Systems, Birmingham,

England, 1977

(57) Spang, H. A., 'Distributed Computer Systems for Control', Preprints of IFAC/IFIP 1st Symposium on Software for Computer Control, Tallinn, USSR, May 1976

(58) Smith, C. L., ' Digital Computer Process Control', Intext 1972

(59) Washburn, J., 'Communications Interface Primer - Part I', Instruments & Control Systems, March 1978

(60) Washburn, J., 'Communications Interface Primer - Part II', Instruments & Control Systems, April 1978

(61) Watkins, D. C. J., 'Distributed Processing - A Logical Approach to Real-Time, On-Line Systems', Proc. of IEE Int. Conf. on Distributed Computer Control Systems, Birmingham, England, Sept. 1977

(62) Watson, J. M., 'Total Distributed Control', Chemical Engineer (UK), March 1976

(63) Wells, A.V., 'TOPSY -A Portable Real-Time Executive for Micro-Processors in Control', Proc. IEE Int. Conf. on Distributed Computer Control Systems,

Birmingham, England, Sept. 1977

(64) Williams, T. J., 'Needs and Desires for a Standardized Real-Time Programming Language', Preprints of IFAC/IFIP 1st Symposium on Software for Computer Control, Tallinn, USSR, May 1976

(65) Wirth, N., 'On the Composition of Well-Structured Programs', Computing Surveys, Vol. 6, No. 4, Dec. 1974

(66) Wo, Y.K., 'A Versatile Microcomputer-Based Controller with an Instrumentation Bus Interface for Data-Acquisition in Automated Experimentation', IEE Trans. Industrial Electronics and Control Instrumentation, Vol. IECI-25, No. 3, Aug 1978

(67) Wirth, N., 'Towards a Discipline of Real-Time Programming', Communications of the ACM, Vol. 20, No. 8, Aug. 1977

(68) Wood, R. K., 'Improved Control by Application of Advanced Control Techniques', ISA Advances in Instrumentation, Vol. 32, 1977

(69) Yourdon, E., ' Techniques of Program Structure & Design', Prentice Hall, 1975

(70) Zeh, A., 'Reliability of Process Control Computer Languages', Preprints of IFAC/IFIP 1st Symposium on Software for Computer Control, Tallinn, USSR, May 1976

Standards:

(71) IEEE 488, 'Digital Interface for Programmable Instrumentation and Related System Components', 1975

(72) IEEE 583, 'Modular Instrumentation and Digital Interface System', 1975

(73) IEEE 595, 'Serial Highway Interface System', 1976

(74) IEEE 596, 'Parallel Highway Interface System', 1976 p 601

(75) IEEE 683, 'Block Transfers in CAMAC Systems', 1976

(76) ISA-S61.1,'Standard Industrial Computer System Fortran Procedures for Executive Functions, Process Input/Output, and Bit Manipulation', 1976

(77) ISA-S5.1, 'Standard Instrumentation Symbols and Identification', 1968

By Personal Communication:

(78) Dr. B.West, Esso Chemical Canada, Sarnia, Canada, 1977

(79) Mr. J.Hamilton, Esso Chemical Canada, Sarnia, Canada, 1978

(80) Dr. H.Wade, Biles & Associates, Houston, USA, 1978

(81) Mr. R.B.Page, GEC Power Engineering Ltd., Leicester, England, 1978

APPENDIX A

MEMORY-IMAGE LISTINGS OF THE ACTIVITY RECORDS FOR THE EXAMPLE APPLICATION OF CHAPTER 6

A.1 Introduction to Appendix A:

This appendix contains listings of the output from the memory-image building program MBILD. The listings (Figs. A.1a to A.1f) document the configuration and contents of the two activity records used to implement the example application described in Chapter 6. The sequence of memory-images shown in the figures of this appendix corresponds to the sequence of contiguous records loaded into the data-base. Each listing shows descriptions and values of all the data-items in a particular activity header or segment together with an octal memory-image.

OCTAL MEMORY IMAGE BUILDER

TITLE ACTIVITY HEADER

AB?60802

FIELD #	DESCRIPTION	VALUE	WORD #	OCTAL IMAGE
		1	1	000001
1	ACTIVITY ID #	37	2	000045
2	ACTIVITY LENGTH	1	3	000001
3	STATUS : ACTIVITY ENABLE/DISABLE	0	3	000001
4	STATUS : LOCAL WRITE UNSYNCH/SYNCH	1	3	000005
5	STATUS : GLOBAL WRITE UNSYNCH/SYNCH	0	3	000005
6	STATUS : NON-CLOCK INITIATED YES/NO	1	3	000025
7	STATUS : DATA STACK IN USE YES/NO		3	000025
8	---- 3-UNUSED STATUS BITS ----	0	3	000025
9	PROCESSOR NETWORK NODE	3	4	000003
10	# OF SEGMENTS	1	4	000403
11	ACTIVITY TYPE #		5	000000
12	# OF POLLS X (# OF SECS / POLL)	4	5	002000
13	SAMPLING INTERVAL (SECS)	0	6	000000
14	PHASE (SECS)	0	6	000000
15	GROUP OF ACTIVITIES ID #	1	7	000001
16	ACTIVITY INPUT DIMENSION	1	7	000401
17	ACTIVITY OUTPUT DIMENSION			

000000

END OF MEMORY IMAGE BUILDING

Fig. A.1a A Listing of Output from the Memory-Image Builder 'MBILD' for the Example Application of Chapter 6

OCTAL MEMORY IMAGE BUILDER

TITLE PROCESS INPUT SEGMENT A3700731

FIELD #	DESCRIPTION	VALUE	WORD #	OCTAL IMAGE
1	SEGMENT ID #	1	1	000001
2	SEGMENT LENGTH	10	1	000001
3	SEGMENT ENABLE/DISABLE	1	2	000001
4	INPUT OPERATIONAL/NON-OP	1	2	000003
5	INPUT FROM PROCESS INTERFACE YES/NO	1	2	000007
6	INPUT FROM BUFFER YES/NO	0	2	000007
7	ZERO	1.22005	3	047025 046402
8	SPAN	0.0006621	5	053310 062355
9	LABEL	SALS	7	043501 046123

INPUT RAW VALUE VECTOR

10	INPUT RAW VALUE		9	000000

INPUT ORIGIN ADDRESS

11	ADDRESS	258	10	000402

END OF MEMORY IMAGE BUILDING 666666

Fig. A.1b A Listing of Output from the Memory-Image Builder 'MBILD' for the Example Application of Chapter 6 ...contd.

OCTAL MEMORY IMAGE BUILDER

TITLE PROCESS CONTROL SEGMENT AB781006

FIELD #	DESCRIPTION	VALUE	WORD #	OCTAL IMAGE
1	SEGMENT ID #	3	1	000003
2	SEGMENT LENGTH	7	1	003403
3	CONTROL ENABLE/DISABLE	1	2	000001
4	LOOP STARTUP YES/NO	1	2	000003
5	RESET WINDUP YES/NO	0	2	000003
6	ALG. IS SUBROUTINE/PROGRAM	1	2	000013
7	SETPOINT CHANGE PERMITTED YES/NO	1	2	000033
8	SPARE 3 BITS		2	000033
9	ALGORITHM #	1	2	000433
10	SETPOINT	0	3	000000
11	PROPORTIONAL CONSTANT	-10	4	177766
12	INTEGRAL TIME	20	5	000024
13	DERIVATIVE TIME	0	6	000000
14	INTEGRAL CONTINUED SUMMATION		7	000000

INPUT POINTER (LENGTH=0 IF STACK ELSE =1)

OUTPUT POINTER (LENGTH=0 IF STACK ELSE=1)

END OF MEMORY IMAGE BUILDING 000000

Fig. A.1c A Listing of Output from the Memory-Image Builder 'MBILD' for the Example Application of Chapter 6 ...contd.

OCTAL MEMORY IMAGE BUILDER

TITLE PROCESS OUTPUT SEGMENT AB780731

FIELD #	DESCRIPTION	VALUE	WORD #	OCTAL IMAGE
1	SEGMENT ID #	2	1	000002
2	SEGMENT LENGTH	13	1	006402
3	SEGMENT ENABLED/DISABLED	1	2	000001
4	OUTPUT OPERATIONAL/NONOP	1	2	000003
5	OUTPUT POSITIONAL/INCREMENTAL	1	2	000007
6	OUTPUT TO PROCESS INTERFACE YES NO	1	2	000017
7	OUTPUT TO BUFFER YES/NO	0	2	000017
8	OUTPUT ON MANUAL YES/NO	0	2	000017
9	OUTPUT TOP LIMIT	32767	3	077777
10	OUTPUT BOTTOM LIMIT	0	4	000000
11	OUTPUT RATE LIMIT	32767	5	077777
12	ZERO		6	040000 000010
13	SPAN		8	040000 000010
14	LABEL		10	020040 020040

OUTPUT VALUE VECTOR

15	OUTPUT VALUE		12	000000

OUTPUT DESTINATION VECTOR

16	OUTPUT ADDRESS	258	13	000402

OUTPUT ORIGIN VECTOR (LENGTH=0 IF STACK)

Fig. A.1d A Listing of Output from the Memory-Image Builder 'MBILD' for the Example Application of Chapter 6 ...contd.

OCTAL MEMORY IMAGE BUILDER

TITLE ACTIVITY HEADER A5760002

FIELD #	DESCRIPTION	VALUE	WORD #	OCTAL IMAGE
1	ACTIVITY ID #	9	1	000011
2	ACTIVITY LENGTH	17	2	000021
3	STATUS : ACTIVITY ENABLE/DISABLE	1	3	000001
4	STATUS : LOCAL WRITE UNSYNCH/SYNCH	0	3	000001
5	STATUS : GLOBAL WRITE UNSYNCH/SYNCH	1	3	000005
6	STATUS : NON-CLOCK INITIATED YES/NO	0	3	000005
7	STATUS : DATA STACK IN USE YES/NO	0	3	000005
8	---- 3-UNUSED STATUS BITS ----	0	3	000005
9	PROCESSOR NETWORK NODE	1	4	000001
10	# OF SEGMENTS	1	4	000401
11	ACTIVITY TYPE #	2	5	001000
12	# OF POLLS * (# OF SECS / POLL)	0	4	000000
13	SAMPLING INTERVAL (SECS)	0	6	000000
14	PHASE (SECS)	1	7	000001
15	GROUP OF ACTIVITIES ID #	0	7	000001
16	ACTIVITY INPUT DIMENSION			
17	ACTIVITY OUTPUT DIMENSION			

END OF MEMORY IMAGE BUILDING 000000

Fig. A.1e A Listing of Output from the Memory-Image Builder 'MBILD' for the Example Application of Chapter 6 ...contd.

OCTAL MEMORY IMAGE BUILDER

TITLE PROCESS INPUT SEGMENT AB780731

FIELD #	DESCRIPTION	VALUE	WORD #	OCTAL IMAGE
1	SEGMENT ID #	1	1	000001
2	SEGMENT LENGTH	10	1	005001
3	SEGMENT ENABLE/DISABLE	1	2	000001
4	INPUT OPERATIONAL/NON-OP	1	2	000003
5	INPUT FROM PROCESS INTERFACE YES/NO	1	2	000007
6	INPUT FROM BUFFER YES/NO	0	2	000007
7	ZERO	-0.44476	3	107044 032777
8	SPAN	0.000904	5	072175 032755
9	LABEL	GPM	7	020107 050115

INPUT RAW VALUE VECTOR

10	INPUT RAW VALUE		9	000000

INPUT ORIGIN ADDRESS

11	ADDRESS	259	10	000403

END OF MEMORY IMAGE BUILDING ******

Fig. A.1f A Listing of Output from the Memory-Image Builder 'MBILD' for the Example Application of Chapter 6 ...contd.

APPENDIX B

A PROPOSED FORMAT FOR A DISCO ARITHMETIC SEGMENT

A.1 Introduction to Appendix B:

This appendix describes a proposed design for a DISCO arithmetic segment such as that required for the decoupling control example of section 6.4. The proposed segment format is shown in table B.1.

Compromises need to be made, when designing such segments and their processors, to ensure that that adequate operational capability is accompanied by reasonable simplicity in processing. It is clearly possible, but in general undesirable, to make the segment so powerful that it constitutes source code to a high-level scientific interpreter. In such a case the segment processor would become the interpreter program.

Proposed processing features for the arithmetic segment are as follows:

(i) Processing would follow a strict sequential order. Parsing of expressions in brackets would not be undertaken. Nested expressions could, however, be

dealt with by storing intermediate results in the segment data area.

(ii) Any number of expressions could be defined in a single segment. The end of an expression would be marked by the '=' operator.

(iii) Integer, logical or floating point modes would be specified by the control byte preceding each expression. Similarly the last expression to be interpreted would be flagged as such by its control byte.

(iv) Operands and results would be specified by pointers in the arithmetic expression.

(v) Typical operators would be :
+, -, X, /, exponentiate, i th root, select greatest, select least, .AND., .OR., .NOT., exclusive .OR.

Table B.2 shows a specific example of the use of the arithmetic segment. This particular segment has been formulated to deal with the arithmetic requirements of the proposed decoupling control scheme of section 6.4.

Building of a segment such as that shown in the table would put to good use a number of features of the data-base

builder. Symbolic handling of the directed labels would allow the user to specify his own names for the operands rather than actual pointers or activity addresses. Facilities for symbolic representation of numerical values provided by the building utilities will permit the use of mnemonic names for constants in the arithmetic operations. In summary, the building of a segment such as the proposed arithmetic segment would be very much like entering equations in FORTRAN source code.

Table B.1 Format of the Proposed Arithmetic Segment

Word	Description	Field Length(bits)
1	Segment ID	8
	Segment length	8
2	Status byte	8
	Control byte	8
3	Address of operand	8
	Operator	8
4	Address of operand	8
•	•	•
•	•	•
•	•	•
	Address of operand	8
	Operator '='	8
	Address for result	8
	Control byte	8
	Address of operand	8
	•	
	•	
	•	
	Address for result	8
	Control byte	8
	DATA AREA	•

Table B.2 Example of Use of the Proposed Arithmetic Segment in the Example of Section 6.4

Word	Description
1	Segment ID Segment length (= 18 words)
2	Status Control byte(floating pt. operation on variables stored in integer format)
3	Address of x +
4	Address of L -
5	Address of L' =
6	Address of Q Control byte(floating pt. operation on variables stored in integer format)
7	Address of x -
8	Address of L +
9	Address of L' =
10	Address of Q' Control byte(floating pt. operation on variables stored in integer format)
11	Address of L +
12	Address of L' /

Table B.2 ...contd.

13	Address of const. =2 =
14	Address of y Control byte (end of expressions)
15	Data (const. =2)
16	Data (Q)
17	Data (Q')
18	Data (y)